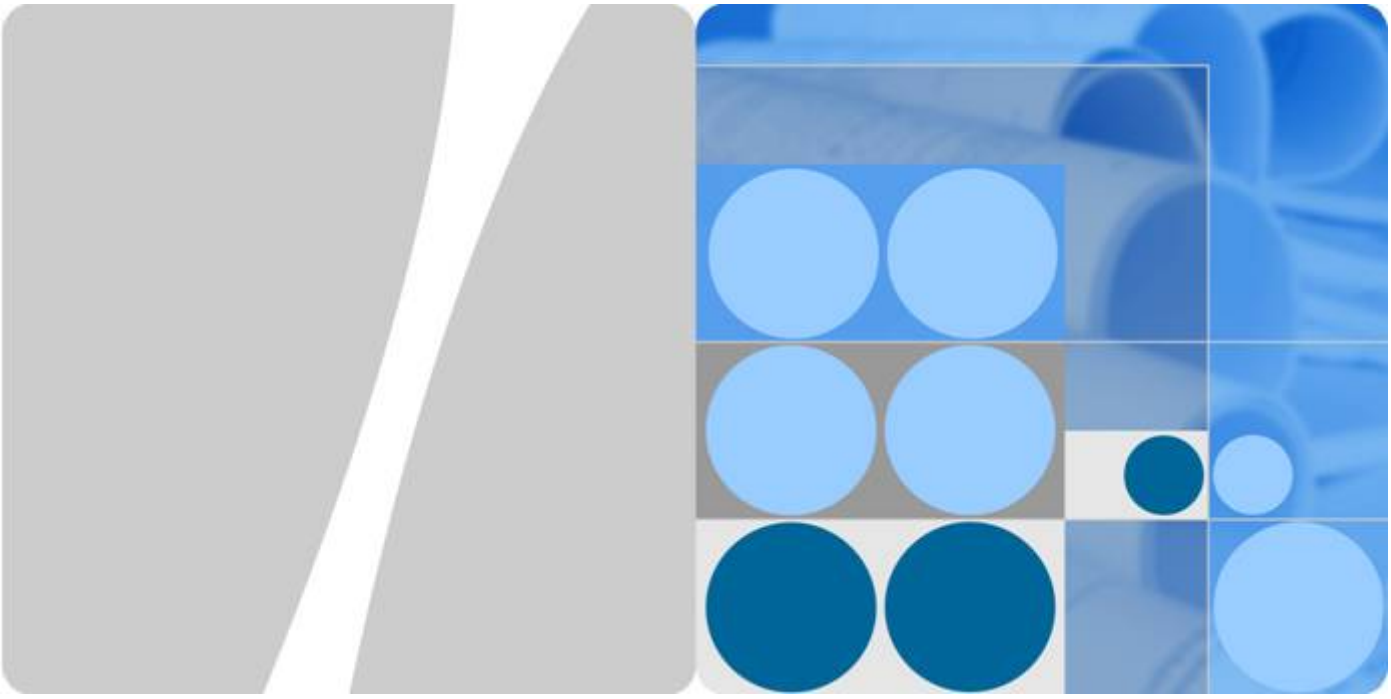

华为 IoT 平台

Agent Lite API 参考(C)

文档版本　01

发布日期　2017-12-07

华为技术有限公司

商标声明


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

注意

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

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

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 前言

导读

本文档系统化描述华为Agent Lite对外开放的能力全集、集成原理和集成参考样例等信息，帮助集成开发者快速而准确的掌握集成方法从而高效实现特定的业务需求。本文档主要包含如下几个部分：

章节	说明	备注
开发者必读	主要是介绍华为Agent Lite开放能力全景图、集成原理、集成流程、集成中关键知识点和常用术语等。确保集成开发者能快速掌握集成的基本知识和技能。	开发者必读章节是必看的。
接口列表	主要是描述能力开放的接口集合，详细介绍每个接口的功能、输入参数、输出参数、和消息样例等信息。	本章节中广播机制介绍是必看的，其他小节根据使用情况查询选看。
常用数据结构定义	主要是给出主要的数据结构、枚举的定义。	-
数据类型定义	统一使用的数据格式定义。	-

修改记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本	发布日期	修改说明	修改影响
01	2016/05/30	第一次发布	

2 开发者必读

1. 概述

华为IoT Agent Lite在智慧家庭、工业物联网、车联网等领域为智能设备提供了标准接入华为IoT联接管理平台的能力。主要面向IPC、轻量级网关、工业网关、车机等计算能力较强的终端/网关设备。

2. 接口全景图

集成开发者通过全景图了解到：Agent Lite提供了哪些功能？每个功能包含哪些接口？接口之间的逻辑关系如何？从而更快速的找到正确的接口来实现具体业务。

功能	接口	说明
直连设备接入	IOTA_Init	初始化Agent Lite模块资源
	IOTA_Destroy	释放Agent Lite模块资源
	IOTA_SetConfig	相关参数配置
	IOTA_Bind	设备绑定
	IOTA_Login	设备登录
	IOTA_DeviceStatusUpdate	设备状态更新
网关管理非直连设备	IOTA_HubDeviceAdd	设备添加
	IOTA_HubDeviceRemove	设备删除
设备服务数据上报	IOTA_ServiceDataReport	设备服务数据上报
设备命令接收	IOTA_TOPIC_SERVICE_COMMAND_ RECEIVE/{deviceId}	设备服务命令接收

3 接口列表

Agent Lite对外提供的接口主要包括广播机制、直连设备的接入、非直连设备的添加和删除、设备数据的上报、设备命令接收以及Json接口。

3.1 广播机制介绍

Agent Lite提供了一套广播机制给第三方开发者，用来接收Agent Lite上报的消息。

订阅广播：

```
HW_BroadCastReg(HW_CHAR *pcTopic ,PFN_HW_BROADCAST_RECV pfnReceiver)
```

广播接收处理函数原型：

```
(*PFN_HW_BROADCAST_RECV)(HW_UINT uiCookie, HW_MSG *pstMsg);
```

此处uiCookie对应于接口中传入的uiCookie，用来匹配业务的请求与响应；如接口中无uiCookie参数，或传入的是无效值，则广播中该参数无意义。

取消订阅广播：

```
HW_BroadCastUnreg(HW_CHAR *pcTopic, PFN_HW_BROADCAST_RECV pfnReceiver);
```

从 pstMsg 获取数据的函数：

获取字符串数据：

```
HW_MsgGetStr(HW_MSG pstMsg, HW_UINT uiTag)
```

获取无符号整形数据：

```
HW_MsgGetUint(HW_MSG pstMsg, HW_UINT uiTag, HW_UINT uiDefault)
```

获取字节数组数据：

```
HW_MsgGetByteArray(HW_MSG pstMsg, HW_UINT uiTag)
```

3.2 直连设备接入

3.2.1 初始化 Agent Lite 资源

接口功能

初始化Agent Lite资源。

接口描述

```
HW_INT IOTA_Init(HW_CHAR *pcWorkPath, HW_CHAR *pcLogPath);
```

参数说明

字段	必选/可选	类型	描述
pcWorkPath	必选	String	Agent Lite工作路径，用于存放Agent Lite的配置文件与生产的临时文件。
pcLogPath	可选	String	日志路径（若日志路径为空则日志写在工作路径中）

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Agent Lite资源
IOTA_Init(“/usr/data”,HW_NULL)
```

3.2.2 释放 Agent Lite 资源

接口功能

调用此函数，Agent Lite会释放申请的所有动态资源（内存、线程等等）。

接口描述

```
IOTA_VOID 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Agent Lite资源
IOTA_Destroy();
```

3.2.3 参数设置

接口功能

配置Agent Lite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	必选	枚举	设备登陆所需的配置项（详见EN_IOTA_CFG_TYPE说明）
pValue	必选	String/int	设置的值

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_CONFIG_IOCM_ADDR, “10.0.0.1”);
IOTA_ConfigSetUint(EN_IOTA_CFG_IOCM_PORT, 8943);
```

3.2.4 设备绑定

接口功能

设备第一次接入IoT联接管理平台时需要进行绑定操作，上层应用通过调用该接口传入设备序列号或者MAC地址以及设备信息来绑定到IoT联接管理平台。

前提条件

在绑定前需要调用**IOTA_ConfigSetXXX接口**设置绑定服务器IP与端口（IoCM服务器地址与端口，Agent Lite会配置默认端口8943），对应参数配置为EN_IOTA_CFG_IOCM_ADDR和EN_IOTA_CFG_IOCM_PORT。

说明

设备绑定是指设备第一次接入IoT平台的过程，需要开发者先在IoT平台注册直连设备，之后在设备上发起绑定操作，将设备绑定到IoT平台上。如果未在IoT平台注册该设备，则绑定操作会失败，Agent Lite将会等待一段时间继续尝试。

接口描述

```
HW_INT IOTA_Bind(HW_CHAR *pcVerifyCode, ST_IOTA_DEVICE_INFO *pstInfo)
```

参数说明

字段	必选/可选	类型	描述
pcVerifyCode	必选	String	设备绑定验证码
pstInfo	必选	ST_IOTA_DEVICE_INFO结构体	设备信息

接口返回值

参见函数标准返回值

说明

此返回值是调用接口的同步返回结果，返回0只是说明接口调用成功，并不说明绑定成功，绑定成功需要收到IOTA_TOPIC_BIND_RSP广播。

注意

该设备未绑定时，该接口会自动尝试重新绑定，如果绑定失败，则30秒后继续进行重试，如果重试超过5次（总计尝试超过6次），则返回失败，不再进行重试。如果想要重新发起绑定，建议让用户重启设备。开发者无需进行重试，如果需要停止重试，调用Agent Lite销毁接口即可。同时请勿短时间内多次调用该接口。

当前绑定流程的重试策略为，如果绑定失败，则30秒后继续进行重试，如果重试超过5次（总计尝试超过6次），则返回失败，不再进行重试。如果想要重新发起绑定，建议让用户重启设备。

返回结果

广播名称	广播参数	成员	描述
IOTA_TOPIC_BIND_RSP	HW_MSG对象	EN_IOTA_BIND_IE_TYPE	绑定返回结果（详见枚举EN_IOTA_BIND_IE_TYPE）

注意事项

如果接入设备是网关设备，则DeviceType填写为“Gateway”,如果为传感器设备直连，则填写为对应的设备类型。

示例

```
//开发者调用该接口进行设备绑定
ST_HW_DEVICE_INFO stDeviceInfo
stDeviceInfo.pcNodeId = “SN Number”;
stDeviceInfo.pcManufacturerId = “Huawei”;
stDeviceInfo.pcDeviceType = “Gateway”;
stDeviceInfo.pcModel = “HW_GW101”;
stDeviceInfo.pcProtocolType = “HuaweiM2M”;

IOTA_Bind(“SN Number”, &stDeviceInfo)
```

当设备成功绑定之后，Agent Lite会返回给UI如下几个参数，需要UI进行持久化存储，设备登录前需要提前进行配置

```
//注册广播接收处理函数
HW_BroadCastReg(“IOTA_TOPIC_BIND_RSP”, Device_RegResultHandler);
```

```
// 开发者注册该函数处理绑定结果
HW_INT Device_RegResultHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
HW_CHAR *pcDeviceId;
HW_CHAR *pcDeviceSecret;
HW_CHAR *pcAppId;
HW_CHAR *pcIoCMServerAddr;
HW_UINT uiIoCMServerPort;
HW_CHAR *pcMqttServerAddr;
HW_UINT uiMqttServerPort;

If (HW_SUCCESS != HW_MsgGetUint(pstMsg, EN_IOTA_BIND_IE_RESULT, 0))
{
Return 0;
}

pcDeviceId = HW_MsgGetStr(pstMsg, EN_IOTA_BIND_IE_DEVICEID);
pcDeviceSecret = HW_MsgGetStr(pstMsg, EN_IOTA_BIND_IE_DEVICESECRET);
pcAppId = HW_MsgGetStr(pstMsg, EN_IOTA_BIND_IE_APPID);
pcIoCMServerAddr = HW_MsgGetStr(pstMsg, EN_IOTA_ BIND_IE_IOCM_ADDR );
uiIoCMServerPort = HW_MsgGetUint(pstMsg, EN_IOTA_BIND_IE_IOCM_PORT, 0);
pcMqttServerAddr = HW_MsgGetStr(pstMsg, EN_IOTA_ BIND_IE_IOCM_ADDR );
uiMqttServerPort = HW_MsgGetUint(pstMsg, EN_IOTA_BIND_IE_IOCM_PORT, 0);

Config_save(“DeviceId”,pcDeviceId);
Config_save(“DeviceSecret”,pcDeviceSecret);
Config_save(“AppId”,pcAppId);
Config_save(“IoCMAddr”,pcIoCMServerAddr);
Config_save(“IoCMPort”,pcIoCMServerPort);
Config_save(“MqttAddr”,pcMqttServerAddr);
Config_save(“MqttPort”,pcMqttServerPort);

return 0;
}
```

3.2.5 设备解绑定

接口功能

注册设备解绑定接收广播来接收处理平台下发的直连设备解绑定命令，开发者收到该广播后需要删除直连设备的配置信息并且释放所有资源，下一次重启后需要重新进行绑定。

接口描述

```
IOTA_TOPIC_CMD_UNBIND_RECEIVE
```

参数说明

无

返回结果

无

示例:

```
// 开发者注册该函数进行解绑定直连设备的处理
HW_INT Gateway_UnbindRecvtHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
```

```
// Delete Config file, free resources.
return 0;
}
//初始化时进行注册该函数
HW_BroadCastReg(“IOTA_TOPIC_CMD_UNBIND_RECEIVE”, Gateway_UnbindRecvtHandler);
```

3.2.6 设备登录

接口功能

设备在第一次绑定后，或者在设备重启后需要进行登录的流程。

接口描述

```
HW_INT IOTA_Login()
```

接口返回值

参见函数标准返回值

说明

此返回值是调用接口的同步返回结果，返回0只是说明接口调用成功，并不说明登陆成功，登陆成功需要收到IOTA_TOPIC_CONNECTED_NTY广播。登录前通过参数配置接口（IOTA_SetConfig）传入所需的登录信息。

必要参数配置:

EN_IOTA_CFG_DEVICEID

EN_IOTA_CFG_DEVICESECRET

EN_IOTA_CFG_APPID

EN_IOTA_CFG_IOCM_ADDR

EN_IOTA_CFG_IOCM_PORT

EN_IOTA_CFG_MQTT_ADDR

EN_IOTA_CFG_MQTT_PORT

之后调用登录接口函数进行登录：

HW_INT IOTA_Login();

返回结果

广播名称	广播参数	成员	描述
IOTA_TOPIC_CONNECTED_NTY	HW_MSG对象	无	登录成功或重连成功
IOTA_TOPIC_DISCONNECT_NTY	HW_MSG对象	EN_IOTA_LGN_IE_TYPE	登陆失败或连接断开

示例：

```
Config_Get(“DeviceId”,pcDeviceId);
Config_Get(“DeviceSecret”,pcDeviceSecret);
```

```
Config_Get(“AppId”,pcAppId);
Config_Get(“HAAddr”,pcHAServerAddr);
Config_Get(“LVSAddr”,pcLVSServerAddr);

IOTA_SetConfig(EN_IOTA_CFG_DEVICEID, pcDeviceId);
IOTA_SetConfig(EN_IOTA_CFG_DEVICESECRET, pcDeviceSecret);
IOTA_SetConfig(EN_IOTA_CFG_APPID, pcAppId);
IOTA_SetConfig(EN_IOTA_CFG_HA_ADDR, pcHAServerAddr);
IOTA_SetConfig(EN_IOTA_CFG_LVS_ADDR, pcLVSServerAddr);

IOTA_Login();
```

然后等待Agent Lite的连接状态广播

需要提前实现连接状态通知广播接收处理函数，建议：

1. 对于网关设备，在连接成功的处理函数中需要进行非直连设备状态上报的处理，并且将缓存的所有上报数据进行上报。
2. 在连接断开的处理函数中记录设备断开状态，之后如果有数据上报，需要进行缓存，等到连接成功后再进行上报。

```
// 开发者注册该函数进行连接成功后的处理
HW_INT Device_ConnectedHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
//update device states
//send buffer data

return 0;
}
```

```
// 开发者注册该函数进行连接失败后的处理
HW_INT Device_DisconnectHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
//stop reporting data
return 0;
}
```

```
//绑定广播接收处理函数
HW_BroadCastReg(“IOTA_TOPIC_CONNECTED_NTY”, Device_ConnectedHandler);
HW_BroadCastReg(“IOTA_TOPIC_DISCONNECT_NTY”, Device_DisconnectHandler);
```

设备登录后，表示该设备已经成功的连接到IoT联接管理平台。

连接成功后，如果因为网络或服务器原因导致连接断开，Agent Lite会自动尝试重新连接，并将实时状态通过这两个广播上报给第三方应用。

3.3 网关管理非直连设备

当开发设备为网关设备时，设备需要管理所有非直连设备（传感器设备）的接入与删除，并且记录这些设备ID与对应设备的映射关系。

3.3.1 设备添加

接口功能

当有新设备接入网关后，通过调用设备添加接口将非直连设备接入IoT联接管理平台，并且获得平台分配的唯一设备逻辑ID。

接口描述

```
HW_INT IOTA_HubDeviceAdd(HW_UINT uiCookie, ST_IOTA_DEVICE_INFO *pstDeviceInfo);
```

参数说明

字段	必选/可选	类型	描述
uiCookie	可选	HW_UINT	Cookie有效值为1-65535
pstDeviceInfo	必选	ST_IOTA_DEVICE_INFO结构体	设备信息

接口返回值

参见函数标准返回值

返回结果

广播名称	广播参数	成员	描述
IOTA_TOPIC_HUB_ADDDEV_RSP	HW_MSG对象	EN_IOTA_HUB_IE_TYPE	返回设备添加结果，如果添加成功则返回设备ID

示例

```
// 开发者调用该接口进行设备添加
ST_IOTA_DEVICE_INFO stDeviceInfo
stDeviceInfo.pcNodeId = “SN Number”;
stDeviceInfo.pcManufacturerId = “Huawei”;
stDeviceInfo.pcDeviceType = “Camera”;
stDeviceInfo.pcModel = “HW_CAM101”;
stDeviceInfo.pcProtocolType = “ONVIF”;
```

IOTA_HubDeviceAdd(29011, &stDeviceInfo)

结果处理：

```
// 开发者注册该函数进行设备添加后的处理
HW_INT Device_AddResultHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
uiResult = HW_MsgGetUint(pstMsg, EN_IOTA_HUB_IE_RESULT);
if (EN_IOTA_HUB_RESULT_SUCCESS != uiResult)
{
// retry with uiCookie
return 0;
}

return 0;
}
```

```
//绑定广播接收处理函数
HW_BroadCastReg(“IOTA_TOPIC_HUB_ADDDEV_RSP”, Device_AddResultHandler);
```

3.3.2 设备状态更新

接口功能

通过该接口更新设备的状态信息，包括直连设备与所管理的非直连设备。设备离线上线均可通过该接口刷新设备状态信息。

说明

直连设备状态通过设备的登录状态进行管理，当直连设备连接断开则表示设备离线，当直连设备连接或重连成功，则表示设备上线，无需通过该接口进行刷新。故建议开发者使用该接口刷新非直连设备的状态。

接口描述

```
HW_INT IOTA_DeviceStatusUpdate(HW_UINT uiCookie, HW_CHAR *pcDeviceId, HW_CHAR pcStatus, HW_CHAR pcStatusDetail)
```

参数说明

字段	必选/可选	类型	描述	
uiCookie	可选	HW_UINT	Cookie有效值为1-65535	
pcDeviceId	必选	HW_CHAR	设备Id	
pcStatus	必选	HW_CHAR	设备状态	在线: ONLINE
				离线: OFFLINE
pcStatusDetail	必选	HW_CHAR	设备状态详细信息	无：NONE
				配置等待：CONFIGURATION_PENDING
				通信错误：COMMUNICATION_ERROR
				配置错误：CONFIGURATION_ERROR
				桥接器离线 BRIDGE_OFFLINE
				固件升级 FIRMWARE_UPDATING
				循环任务 DUTY_CYCLE
				未激活 NOT_ACTIVE

接口返回值

参见函数标准返回值

返回结果

广播名称	广播参数	成员	描述
IOTA_TOPIC_DEVUPDATE_RSP/{deviceId}	HW_MSG对象	无	设备状态更新结果

示例：

```
HW_CHAR *pcDeviceId = stDevice.pcDeviceId;

IOTA_DeviceStatusUpdate(0, pcDeviceId, “ONLINE” , “NONE”);
```

然后等待命令执行结果

```
// 开发者注册该函数进行状态更新后的处理
HW_INT Device_StatusUpdateHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
HW_CHAR pcCmdContent;
pcCmdContent = HW_MsgGetStr(pstMsg, EN_IOTA_DEVUPDATE_IE_RESULT);
pcCmdContent = HW_MsgGetStr(pstMsg, EN_IOTA_DEVUPDATE_IE_DEVICEID);

return 0;
}

//绑定广播接收处理函数
HW_BroadCastReg(“IOTA_TOPIC_DEVUPDATE_RSP”, Device_StatusUpdateHandler);
```

3.3.3 设备删除

接口功能

当有新设备需要从网关移除时，通过调用设备删除接口将非直连设备从IoT联接管理平台删除。

接口描述

```
HW_INT IOTA_HubDeviceRemove(HW_UINT uiCookie, HW_CHAR *pcDeviceId);
```

参数说明

字段	必选/可选	类型	描述
uiCookie	可选	HW_UINT	Cookie有效值1-65535
pcDeviceId	必选	String	设备Id

接口返回值

参见函数标准返回值

返回结果

广播名称	广播参数	成员	描述
IOTA_TOPIC_HUB_RMVDEV_RSP	HW_MSG对象	EN_IOTA_HUB_IE_TYPE	删除结果

示例

```
// 开发者调用该接口进行设备删除
HW_CHAR *pcDeviceId = stDevice.pcDeviceId;

IOTA_HubDeviceRemove(HW_NULL, pcDeviceId);
结果处理：
HW_INT Device_RemoveResultHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
uiResult = HW_MsgGetUint (pstMsg, EN_IOTA_HUB_IE_RESULT);
if (EN_IOTA_HUB_RESULT_SUCCESS != uiResult)
{
// retry with uiCookie
return 0;
}

return 0;
}

HW_BroadCastReg(“IOTA_TOPIC_HUB_RMVDEV_RSP”, Device_RemovResultHandler);
```

3.4 设备服务数据上报

接口功能

当非直连设备上报数据到网关时，网关需要调用设备服务数据上报接口将数据上报到IoT联接管理平台。

接口描述

```
HW_INT IOTA_ServiceDataReport(HW_UINT uiCookie, HW_CHAR *pcRequstId,
HW_CHAR *pcDeviceId, HW_CHAR *pcServiceId, HW_CHAR *pcServiceProperties);
```

参数说明

字段	必选/可选	类型	描述
uiCookie	可选	unsign int	Cookie有效值1-65535
pcRequstId	条件必选	String	请求ID，用来匹配之前平台下发的服务命令。当该次数据上报为此前某一次命令请求的响应时，需要填写此次命令请求的请求ID。

字段	必选/可选	类型	描述
pcDeviceId	必选	String	设备ID
pcServiceId	必选	String	服务ID
pcServiceProperties	必选	String	服务属性

接口返回值

参见函数标准返回值

返回结果

广播名称	广播参数	成员	描述
IOTA_TOPIC_DATATRANS_REPORT_RSP/{deviceId}	HW_MSG对象	EN_IOTA_DATAREPORT_IE_TYPE	数据上报结果

示例

用户根据Profile格式使用Json组件拼装服务属性的内容(pcServiceProperties)

```
HW_UINT *uiLen;

IOTA_ServiceDataReport(1211, NULL, "xxxx_xxxx_xxxx_xxxx"
, "DoorWindow", "{\"status\":\"OPEN\"}");
```

数据上报结果接收

```
//开发者注册该函数进行设备服务数据上报后的处理
HW_INT Device_DataReportResultHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
uiResult = HW_MsgGetUint(pstMsg, EN_IOTA_DATATRANS_IE_RESULT);
if (HW_SUCCESS != uiResult)
{
// retry with uiCookie
return 0;
}

return 0;
}
```

```
//在设备添加成功后立即注册服务数据上报结果接收广播
HW_BroadCastReg("IOTA_TOPIC_SERVICE_REPORT_RET/XXXX_XXXX_XXXX_XXXX", Device_AddResultHandler);
```

3.5 设备命令接收

接口功能

注册设备命令接收广播来接收处理平台下发的控制命令。

接口描述

```
IOTA_TOPIC_SERVICE_COMMAND_RECEIVE/{deviceId}
```

参数说明

请参考**EN_IOTA_DATATRANS_IE_TYPE 消息信元枚举说明**

返回结果

无

示例

```
// 开发者注册该函数进行命令接收的处理
HW_INT Switch_CommandRecvtHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
HW_CHAR *pcMethod, *pcServiceId, *pcCmdContent, *pcDeviceId;

pcDeviceId = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_DEVICEID);
pcServiceId = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_SERVICEID);
pcMethod = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_METHOD);
pcCmdContent = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_CMDCONTENT);

if (strcmp(pcServiceId, “switch”))
{
//根据Proflie定义的命令参数，使用Json组件解析pcCmdContent
//Send command to Switch
}

return 0;
}
```

```
//在设备添加成功后立即注册设备命令接收广播
HW_BroadCastReg(“IOTA_TOPIC_SERVICE_CMD_RECEIVE/XXXX_XXXX_XXXX_XXXX”,
Device_AddResultHandler);
```

开发者需要在设备添加成功后注册该设备的命令接收广播，广播主题为

“ IOTA_TOPIC_SERVICE_CMD_RECEIVE/设备ID”，Agent Lite收到平台发往给设备的命令后会直接广播给该设备注册的广播处理函数。如果开发者不需要按设备进行分发，直接使用主题名即可即可，即” IOTA_TOPIC_SERVICE_CMD_RECEIVE”。

3.6 Json 组件使用说明

该组件为Agent Lite提供给开发者的工具组件，如果开发者无法进行Json格式的编码和解码，则可以使用该组件进行编码和解码。主要用于上报数据组装与下发命令解析。

1. Json 编码

使用Json组件进行编码的流程。

创建Json编码对象。

```
HW_JSONOBJ HW_JsonObjCreate()
```

获取Json对象根节点。

```
HW_JSON HW_JsonGetJson(HW_JSONOBJ hjson)
```

往Json对象中添加键值对：

添加pcVal为字符串的Json键值对。

```
HW_INT HW_JsonAddStr(HW_JSON *pstJson, HW_CHAR *pcKey, HW_CHAR *pcVal)
```

添加uiVal为整数的Json键值对。

```
HW_INT HW_JsonAddUint(HW_JSON *pstJson, HW_CHAR *pcKey, HW_UINT uiVal)
```

添加bVal为bool的Json键值对。

```
HW_INT HW_JsonAddBool(HW_JSON *pstJson, HW_CHAR *pcKey, HW_BOOL bVal)
```

添加值为Json的Json键值对，获取到的为子Json对象。

```
HW_JSON HW_JsonAddJson(HW_JSON *pstJson, HW_CHAR *pcKey)
```

添加值为Json数组Json键值对，获取到的为子Json数组对象

```
HW_JSON_ARRAY HW_JsonAddJsonArray(HW_JSON *pstJson, HW_CHAR *pcKey)
```

往Json数组中添加键值对：

添加pcVal为字符串的Json键值对。

```
HW_INT HW_JsonArrayAddStr(HW_JSON_ARRAY *pstArray, HW_CHAR *pcKey, HW_CHAR *pcVal)
```

添加uiVal为整数的Json键值对。

```
HW_INT HW_JsonArrayAddUint(HW_JSON_ARRAY *pstArray, HW_CHAR *pcKey, HW_UINT uiVal)
```

添加bVal为bool的Json键值对。

```
HW_INT HW_JsonArrayAddBool(HW_JSON_ARRAY *pstArray, HW_CHAR *pcKey, HW_BOOL bVal)
```

添加pucValue为Json的Json值，获取到的为子Json对象。

```
HW_JSON HW_JsonArrayAddJson(HW_JSON_ARRAY *pstArray)
```

添加pucValue为Json数组Json键值对，获取到的为子Json数组对象

```
HW_JSON_ARRAY HW_JsonArrayAddJsonArray(HW_JSON_ARRAY *pstArray, HW_CHAR *pcKey)
```

获取Json字符串

```
HW_CHAR HW_JsonEncodeStr(HW_JSONOBJ hJson);
```

删除Json对象

```
HW_VOID HW_JsonObjDelete(HW_JSONOBJ *phJson);
```

Json编码示例：

待解析Json格式：

```
{
"temperature":22
"otherInfo":
```

```
{
“batteryLevel”:”low”
}
}
/*变量定义*/
HW_JSONOBJ jsonObj;
HW_JSON rootjson;
HW_JSON json;
HW_CHAR *pcJsonStr;

/*创建Json编码对象*/
hJsonObj = HW_JsonObjCreate();

/*获取跟节点Json对象*/
rootjson = HW_JsonGetJson(hJsonObj);

/*往根节点中添加键值对*/
HW_JsonAddUint(rootjson, "temperature", 22);

/*从根节点中获取子Json对象*/
json = HW_JsonAddJson(rootjson, "otherInfo");

/*在子Json中添加键值对*/
HW_JsonAddStr(json, " batteryLevel", "low");


/*获取Json字符串*/
pcJsonStr = HW_JsonEncodeStr(hjsonObj);

/*删除之前创建的Json编码对象，释放资源*/
HW_JsonObjDelete(&hJsonObj);
```

2. Json 解码

使用Json组件进行解码的流程

创建Json解析对象。

```
HW_JSONOBJ HW_JsonDecodeCreate(HW_CHAR *pucStr, HW_BOOL bStrCpy)
```

获取Json解析对象中的Json数据部分。

```
HW_JSON HW_JsonGetJson(HW_JSONOBJ hJson)
```

获取Json数据中与pucKey对应的字符串。

```
HW_CHAR *HW_JsonGetStr(HW_JSON pstJson, HW_CHAR *pucKey)
```

获取Json数据中与pucKey对应的无符号整型。

```
HW_UINT HW_JsonGetUint(HW_JSON pstJson, HW_CHAR *pucKey, HW_UINT uiDft)
```

获取Json数据中与pucKey对应的Boolean值。

```
HW_BOOL HW_JsonGetBool(HW_JSON pstJson, HW_CHAR *pucKey, HW_BOOL bDft)
```

获取Json数据中与pucKey对应的数组。

```
HW_UJSON_ARRAY HW_JsonGetArray(HW_JSON pstJson, HW_CHAR *pucKey)
```

获取Json数组的长度。

```
HW_UINT HW_JsonArrayGetCount(HW_UJSON_ARRAY pstArray)
```

获取Json数组中序号为uiIndex项的Json数据。

```
HW_JSON HW_JsonArrayGetJson(HW_UJSON_ARRAY pstArray, HW_UINT uiIndex)
```

获取Json数组中序号为uiIndex项的无符号整形。

```
HW_UINT HW_JsonArrayGetUint(HW_UJSON_ARRAY pstArray, HW_UINT uiIndex, HW_UINT uiDft)
```

获取Json数组中序号为uiIndex项的Boolean值。

```
HW_UINT HW_JsonArrayGetBool(HW_UJSON_ARRAY pstArray, HW_UINT uiIndex, HW_BOOL bDft)
```

获取Json数组中序号为uiIndex项的字符串。

```
HW_CHAR *HW_JsonArrayGetStr(HW_UJSON_ARRAY pstArray, HW_UINT uiIndex)
```

获取Json数组中序号为uiIndex项的子数组。

```
HW_UJSON_ARRAY HW_JsonArrayGetArray(HW_UJSON_ARRAY pstArray, HW_UINT uiIndex)
```

删除之前创建的Json解析对象。

```
HW_VOID HW_JsonObjDelete(HW_JSONOBJ *phJson)
```

Json解析示例：

待解析Json格式：

```
{
"action": "notify",
"type": "userstate",
"userstateinfo":
[
{ "num": "11111 ", "state": "idle"},
{ "num": "11111", "state": "ringing"}
]
}
/*变量定义*/
HW_JSONOBJ jsonObj;
HW_JSON json;
HW_UJSON_ARRAY jsonArray;
HW_CHAR *action;
HW_CHAR *type;
HW_UINT count;
HW_UINT index;

/*创建Json解析对象*/
jsonObj = HW_JsonDecodeCreate(jsonStr, HW_TRUE);

/*获取Json解析对象中的Json数据部分*/
json = HW_JsonGetJson(jsonObj);

/*获取Json数据中与"action"对应的字符串*/
action = HW_JsonGetStr(json, "action");

/*获取Json数据中与"type"对应的字符串*/
type = HW_JsonGetStr(json, "type");

/*获取Json数据中与"userstateinfo"对应的Json数组*/
jsonArray = HW_JsonGetArray(json, "userstateinfo");

/*获取数组jsonArray的长度*/
count = HW_JsonArrayGetCount(jsonArray);
for (index = 0; index < count; index++)
```

```
{
/*获取数组jsonArray中序号为index项的Json数据*/
HW_JSON jsonItem = HW_JsonArrayGetJson(jsonArray, index);

/*获取jsonItem中与" num "对应的字符串*/
HW_CHAR *num = HW_JsonGetStr(jsonItem, "num");

/*获取jsonItem中与" state "对应的字符串*/
HW_CHAR *state = HW_JsonGetStr(jsonItem, "state");
......
}

/*删除之前创建的Json解析对象，释放资源*/
HW_JsonObjDelete(jsonObj);
```

4 常用数据结构定义

1. ST_IOTA_DEVICE_INFO 结构体说明

字段	必选/可选	类型	描述
pcNodeId	必选	String	关键参数，对接平台的网关下设备唯一标识，设备填写，平台用于判重
pcName	可选	String	设备名称
pcDescription	可选	String	设备描述
pcManufacturerId	必选	String	厂商ID
pcManufacturerName	可选	String	厂商名
pcMac	可选	String	设备MAC地址
pcLocation	可选	String	设备的位置
pcDeviceType	必选	String	设备类型
pcModel	必选	String	型号 z-wave: ProductType + ProductId 16 进制： XXXX-XXXX 补0对齐 如：001A-0A12 其他协议再定
pcSwVersion	可选	String	软件版本， Z-Wave 主次版本号 主版本号.次版本号 如：1.1

字段	必选/可选	类型	描述
pcFwVersion	可选	String	固件版本
pcHwVersion	可选	String	硬件版本
pcProtocolType	必选	String	协议类型 Z-Wave
pcBridgeId	可选	String	表示设备通过哪个Bridge接入平台
pcStatus	可选	String	表示设备是否在线， ONLINE 在线 OFFLINE 离线
statusDetail	可选	String	状态详情，如果pcStatus不为空，则该参数必选。 参数值： 无： NONE 配置等待： CONFIGURATION_PENDING 通信错误： COMMUNICATION_ERROR 配置错误： CONFIGURATION_ERROR 桥接器离线 BRIDGE_OFFLINE 固件升级 FIRMWARE_UPDATING 循环任务 DUTY_CYCLE 未激活 NOT_ACTIVE
pcMute	可选	String	表示设备是否被屏蔽： TRUE FALSE

2. EN_IOTA_BIND_IE_TYPE 消息信元枚举说明

枚举项	枚举值	类型	描述
EN_IOTA_BIND_IE_RESULT	0	EN_IOTA_BIND_RESULT_TYPE	绑定结果
EN_IOTA_BIND_IE_DEVICEID	1	String	平台分配的逻辑设备ID
EN_IOTA_BIND_IE_DEVICESECRET	2	String	设备接入的鉴权秘钥
EN_IOTA_BIND_IE_APPID	3	String	开发者应用ID
EN_IOTA_BIND_IE_IOCM_ADDR	4	String	IoCM服务器地址
EN_IOTA_BIND_IE_IOCM_PORT	5	unsigned int	IoCM服务器端口
EN_IOTA_BIND_IE_MQTT_ADDR	6	String	MQTT服务器地址
EN_IOTA_BIND_IE_MQTT_PORT	7	unsigned int	MQTT服务器端口
EN_IOTA_BIND_IE_IODM_ADDR	8	String	IoDM服务器地址
EN_IOTA_BIND_IE_IODM_PORT	9	unsigned int	IoDM服务器端口

3. EN_IOTA_BIND_RESULT_TYPE 参数枚举说明

枚举项	枚举值	类型	描述
EN_IOTA_BIND_RESULT_SUCCESS	0	NA	绑定成功
EN_IOTA_BIND_RESULT_DEV_NOT_BIND	1	NA	未扫码
EN_IOTA_BIND_RESULT_VERIFYCODE_EXPIRED	2	NA	验证码过期
EN_IOTA_BIND_RESULT_FAILED	255	NA	其余失败

4. EN_IOTA_LGN_IE_TYPE 消息信元枚举说明

枚举项	枚举值	类型	描述
EN_IOTA_LGN_IE_REASON	0	EN_IOTA_LGN_REASON_TYPE	登录失败原因

5. EN_IOTA_LGN_REASON_TYPE 参数枚举说明

枚举项	枚举值	类型	描述
EN_IOTA_LGN_REASON_NULL	0	NA	无原因
EN_IOTA_LGN_REASON_CONNCET_ERR	1	NA	连接失败
EN_IOTA_LGN_REASON_SERVER_BUSY	2	NA	服务器忙
EN_IOTA_LGN_REASON_AUTH_FAILED	3	NA	鉴权失败、开发者需要停止重新尝试登陆。
EN_IOTA_LGN_REASON_NET_UNAVAILABLE	5	NA	网络不可用
EN_IOTA_LGN_REASON_DEVICE_NOEXIST	12	NA	设备不存在、开发者需要停止重新尝试登陆。
EN_IOTA_LGN_REASON_DEVICE_RMVED	13	NA	设备已删除、开发者需要停止重新尝试登陆。
EN_IOTA_LGN_REASON_UNKNOWN	255	NA	未知原因

6. EN_IOTA_CFG_TYPE 参数枚举说明

枚举项	枚举值	类型	描述
EN_IOTA_CFG_DEVICEID	0	String	平台分配的逻辑设备ID
EN_IOTA_CFG_DEVICESECRET	1	String	设备接入的鉴权秘钥

枚举项	枚举值	类型	描述
EN_IOTA_CFG_APPID	2	String	开发者应用ID
EN_IOTA_CFG_IOCM_ADDR	3	String	IoCM服务器地址
EN_IOTA_CFG_IOCM_PORT	4	unsigned int	IoCM服务器端口
EN_IOTA_CFG_MQTT_ADDR	5	String	MQTT服务器地址
EN_IOTA_CFG_MQTT_PORT	**6**	unsigned int	MQTT服务器端口
EN_IOTA_CFG_IODM_ADDR	**7**	String	IoDM服务器地址
EN_IOTA_CFG_IODM_PORT	**8**	unsigned int	IoDM服务器端口

7. EN_IOTA_HUB_IE_TYPE 消息信元枚举说明

枚举项	枚举值	类型	描述
EN_IOTA_HUB_IE_RESULT	0	EN_IOTA_HUB_RESULT_TYPE	添加/删除执行结果
EN_IOTA_HUB_IE_DEVICEID	1	String	添加成功后分配的设备ID

8. EN_IOTA_HUB_RESULT_TYPE 参数枚举说明

枚举项	枚举值	类型	描述
EN_IOTA_HUB_RESULT_SUCCESS	0	NA	添加/删除执行成功
EN_IOTA_HUB_RESULT_DEVICE_EXIST	1	NA	设备已存在
EN_IOTA_HUB_RESULT_DEVICE_NOTEXIST	2	NA	设备不存在
EN_IOTA_HUB_RESULT_DEVICE_FAILED	255	NA	执行失败

9. EN_IOTA_DATATRANS_IE_TYPE 消息信元枚举说明

枚举项	枚举值	类型	描述	
EN_IOTA_DATATRANS_IE_RESULT	0	unsigned int	命令执行返回结果	成功：0
				失败：1
EN_IOTA_DATATRANS_IE_DEVICEID	1	String	设备ID	
EN_IOTA_DATATRANS_IE_REQUESTID	2	String	请求ID	
EN_IOTA_DATATRANS_IE_SERVICEID	3	String	服务ID	
EN_IOTA_DATATRANS_IE_METHOD	4	String	服务方法	
EN_IOTA_DATATRANS_IE_CMDCONTENT	5	String	命令内容，以Json格式进行拼装的服务命令参数，开发者根据对应服务命令的定义进行Json解析，从而获取命令参数值。	

10. EN_IOTA_DEVUPDATE_IE_TYPE 消息信元枚举说明

枚举项	枚举值	类型	描述	
EN_IOTA_DEVUPDATE_IE_RESULT	0	unsigned int	命令执行返回结果	成功：0
				失败：1
EN_IOTA_DEVUPDATE_IE_DEVICEID	**1**	**String**	设备ID	

5 数据类型定义

1. 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

2. 函数标准返回值

返回值名称	值	类型原型
HW_OK	0	执行成功
HW_ERR	1	执行错误
HW_ERR_PTR	2	错误的指针
HW_ERR_ID	3	错误的ID
HW_ERR_PARA	4	错误的参数
HW_ERR_KEY	5	错误的KEY

返回值名称	值	类型原型
HW_ERR_NOMEM	6	内存不足
HW_ERR_MAGIC	7	保留
HW_ERR_OVERFLOW	8	存在溢出
HW_ERR_GVAR	9	保留
HW_ERR_POOL	10	保留
HW_ERR_NO_MUTEX	11	未加锁
HW_ERR_PID	12	保留
HW_ERR_FILEOPEN	13	文件打开失败
HW_ERR_FD	14	错误的文件描述符
HW_ERR_SOCKET	15	SOCKET异常
HW_ERR_NOTSUPPORT	16	不支持
HW_ERR_NOTLOAD	17	未加载
HW_ERR_ENCODE	18	编码错误
HW_ERR_DECODE	19	解码错误
HW_ERR_CALLBACK	22	错误的回调函数
HW_ERR_STATE	23	错误的状态
HW_ERR_OVERTIMES	24	重试超过次数
HW_ERR_ENDOVER	25	保留
HW_ERR_ENDLINE	26	保留
HW_ERR_NUMBER	27	错误的数字
HW_ERR_NOMATCH	28	不匹配
HW_ERR_NOSTART	29	未开始
HW_ERR_NOEND	30	未结束
HW_ERR_OVERLAP	31	保留
HW_ERR_DROP	32	丢弃
HW_ERR_NODATA	33	无数据
HW_ERR_CRC_CHK	34	CRC校验失败
HW_ERR_AUTH	35	鉴权失败
HW_ERR_LENGTH	36	长度错误
HW_ERR_NOTALLOW	37	不被允许的操作

返回值名称	值	类型原型
HW_ERR_TOKEN	38	凭据错误
HW_ERR_NOTIPV4	39	不支持IPV4
HW_ERR_NOTIPV6	40	不支持IPV6
HW_ERR_IELOST	41	保留
HW_ERR_IELOST1	42	保留
HW_ERR_IELOST2	43	保留
HW_ERR_AUDIO	44	保留
HW_ERR_VIDEO	45	保留
HW_ERR_MD5	46	保留
HW_ERR_MD5_HA1	47	保留
HW_ERR_MD5_RES	48	保留
HW_ERR_DIALOG	49	错误的对话
HW_ERR_OBJ	50	错误的对象