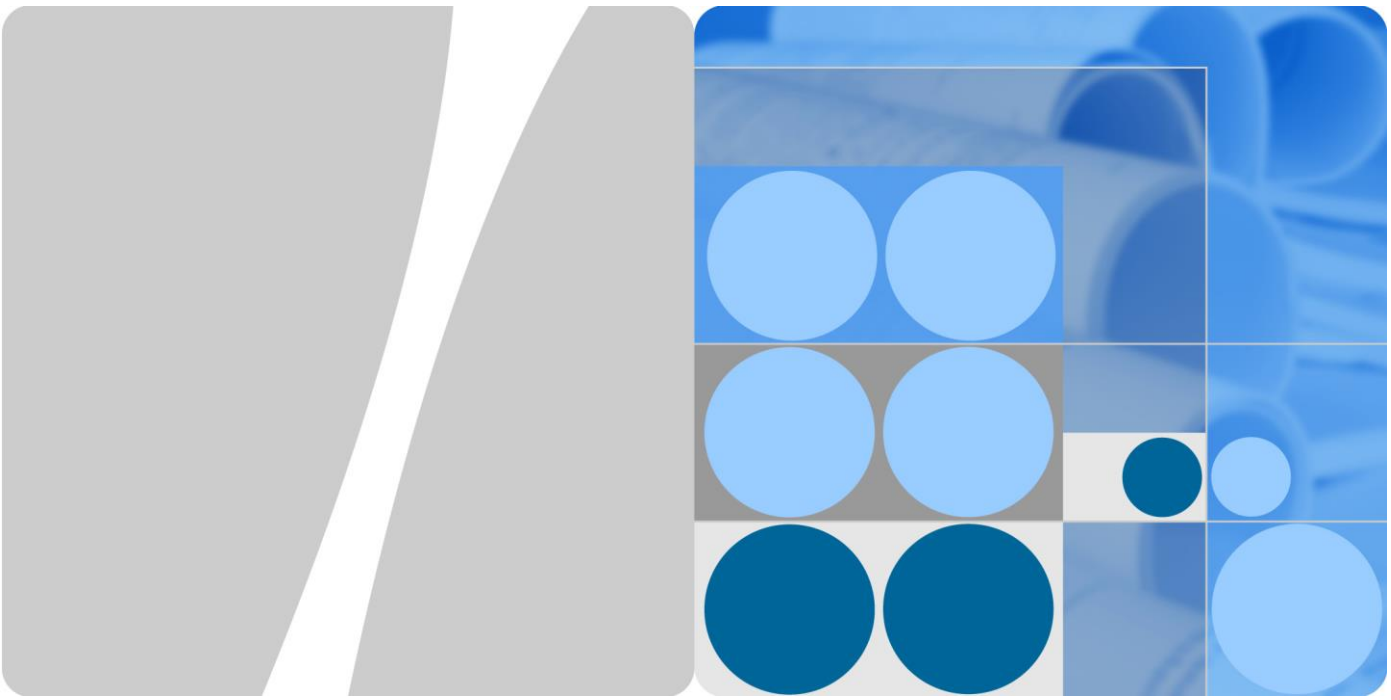

02 IoT Device Access IoT Device SDK API Reference (C) Official

Issue

Date 2023-05-30

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

Contents

1 Before You Start

Introduction

The IoT platform provides the IoT Device SDK (SDK for short) for fast device access. After being integrated with the SDK, devices that support the TCP/IP protocol stack can directly communicate with the platform. Devices that do not support the TCP/IP protocol stack, such as Bluetooth and Zigbee devices, can forward device data to the platform through gateways integrated with the SDK.

API List

The table below lists the APIs provided by the SDK.

- Directly connected devices can be connected to the platform after getting authenticated.
- Indirectly connected devices can be connected to the platform through gateways.

Function	API	Description
Access of directly connected devices	IOTA_Init	3.1 Initializing SDK Resources
	IOTA_Destroy	3.2 Releasing SDK Resources
	IOTA_ConfigSetXXX /	3.3 Configuring a Binding Between the Platform and Device
	IOTA_SetPrintLogCallback	3.4 Setting Log Printing Callback Functions
	IOTA_SetCallbackXXX	3.5 Setting Callback Functions
	IOTA_Connect	3.6 Connecting to the Platform
	IOTA_DisConnect	3.7 Disconnecting from the Platform
Data Reporting from Directly	IOTA_MessageReport	4.1 Reporting a Message to a Custom Topic

Function	API	Description
Connected Devices	IOTA_PropertiesReport	4.2 Reporting Properties
	IOTA_CommandResponse	4.3 Reporting the Command Execution Result
	IOTA_PropertiesSetResponse	4.4 Reporting the Property Setting Result
	IOTA_PropertiesGetResponse	4.5 Reporting the Property Query Result
	IOTA_UpdateSubDeviceStatus	4.6 Reporting the Child Device Status
	IOTA_AddSubDevice	4.7 Requesting for Adding a Child Device
	IOTA_DelSubDevice	4.8 Requesting for Deleting a Child Device
Topic Processing	HW_API_FUNC HW_INT IOTA_SubscribeUserTopic(HW_CHAR *topicParas)	5.2.1 Subscribing to a Custom Topic
	HW_API_FUNC HW_VOID IOTA_SetUserTopicMsgCallback(PFN_USER_TOPIC_MSG_CALLBACK_HANDLER pfnCallbackHandler)	5.2.2 Receiving a Message from a Custom Topic
	HW_API_FUNC HW_INT IOTA_SubscribeBoostrap()	5.2.3 Subscribing to a Bootstrap Topic
Command receipt for directly connected devices (callback APIs)	HW_VOID (*PFN_MESSAGE_CALLBACK_HANDLER)(EN_IOTA_MESSAGE *message)	5.1 Receiving a Message
	HW_VOID (*PFN_CMD_CALLBACK_HANDLER)(EN_IOTA_COMMAND *message)	5.3 Receiving a Command
	HW_VOID (*PFN_PROP_SET_CALLBACK_HANDLER)(EN_IOTA_PROPERTY_SET *message)	5.4 Receiving a Device Property Setting Request
	HW_VOID (*PFN_PROP_GET_CALLBACK_HANDLER)(EN_IOTA_PROPERTY_GET *message)	5.5 Receiving a Device Property Query Request
	HW_VOID (*PFN_EVENT_CALLBACK_HANDLER)(EN_IOTA_EVENT	5.6 Receiving a Child Device Addition Notification

Function	API	Description
	*message)	
	HW_VOID (*PFN_EVENT_CALLBACK_HANDLER)(EN_IOTA_EVENT *message)	5.7 Receiving a Child Device Deletion Notification
Data reporting from child devices	IOTA_MessageReport	6.1 Reporting a Message
	IOTA_BatchPropertiesReport	6.2 Reporting Device Properties in Batches
OTA upgrade	HW_VOID (*PFN_CALLBACK_HANDLER)	7.1 Platform Requesting the Software or Firmware Version
	IOTA_OTAVersionReport	7.2 Device Reporting the Software or Firmware Version
	HW_VOID (*PFN_CALLBACK_HANDLER)	7.3 Platform Delivering an Upgrade Notification
	IOTA_GetOTAPackages	7.4.1 Downloading an OTA Package into the Current Directory
	IOTA_GetOTAPackages_Ext	7.4.2 Downloading an OTA Package into a Specific Path
	IOTA_OTAStatusReport	7.5 Device Reporting the Upgrade Status
Device shadow	IOTA_GetDeviceShadow	8.1 Requesting Shadow Data
	HW_VOID (*PFN_CALLBACK_HANDLER_WITH_TOPIC)	8.2 Receiving Shadow Data
Device Time Synchronization	IOTA_GetNTPTime	9.1 Requesting for Time Synchronization
Device Reporting Information	IOTA_ReportDeviceInfo	10.1 Reporting Information
Device Reporting Logs	IOTA_ReportDeviceLog	11.2 Reporting Logs to the Platform
Remote SSH Login	IOTA_RptDeviceConfigRst	12.1 Reporting the Remote SSH Configuration Result
Device-side Rules	IOTA_EnableDeviceRuleStorage	13.1 Storing Device-side Rules Locally
M2M Communication	IOTA_M2MSendMsg	14.1 Sending M2M Messages
Device-Cloud	IOTA_GetLatestSoftBusInfo	15.1 Requesting for the

Function	API	Description
Security Component		Latest Soft Bus Information

2 Instructions Before Development

2.1 Types

Typical Types

Type Name	Standard Type Name
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

Function Return Values

Return Value Name	Value	Description
IOTA_SUCCESS	0	Execution succeeded.
IOTA_FAILURE	-1	An execution error occurred.
IOTA_PARAMETER_EMPTY	-101	Some parameters are empty.

Return Value Name	Value	Description
IOTA_RESOURCE_NOT_AVAILABLE	-102	The resource is not allowed.
IOTA_INITIALIZATION_REPEATED	-103	The resource cannot be re-initialized.
IOTA_LIBRARY_LOAD_FAILED	-104	Failed to load the library file.
IOTA_SECRET_ENCRYPT_FAILED	-105	Failed to encrypt the device secret.
IOTA_MQTT_CONNECT_FAILED	-106	A connection error occurred.
IOTA_MQTT_CONNECT_EXISTED	-107	The connection already exists.
IOTA_CERTIFICATE_NOT_FOUND	-108	The certificate is not found.
IOTA_MQTT_DISCONNECT_FAILED	-109	Disconnection failed.
IOTA_PARSE_JSON_FAILED	-110	Failed to parse the string.
IOTA_PARAMETER_ERROR	-111	Invalid parameters.
IOTA_NUMBER_EXCEEDS	-112	The number exceeds the upper limit.

3 Access of Directly Connected Devices

3.1 Initializing SDK Resources

API Function

This API is used to initialize SDK resources.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
pcWorkPath	Mandatory	HW_CHAR*	SDK working path, which is used to store the SDK configuration file. The working path must be valid and end with a null-terminated string (\0). You are advised to set this parameter to ***.(current path)***. The default certificate file is in the **conf** folder in the current path.

Return Value

For details, see Function Return Values.

Example

```
// Call this API to initialize SDK resources.
IOTA_Init("."); // Initialize the SDK resources in the current path.
```

3.2 Releasing SDK Resources

API Function

This API is used to release all dynamic resources that have been applied for, such as the memory and threads.

API Description

```
HW_INT IOTA_Destroy()
```

Return Value

For details, see Function Return Values.

Example

```
// Call this API to release SDK resources.
IOTA_Destroy();
```

3.3 Configuring a Binding Between the Platform and Device

API Function

This API is used to configure SDK parameters to bind a device with the platform.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
iItem (key)	Mandatory	HW_INT	Configuration items used for device binding. • **EN_IOTA_CFG_DEVICEID**: used to set

Parameter	Mandatory or Optional	Type	Description
			the device ID. • **EN_IOTA_CFG_DEVICESECRET**: used to set the device secret. • **EN_IOTA_CFG_MQTT_ADDR**: used to set the platform IP address. • **EN_IOTA_CFG_MQTT_PORT**: used to set the platform port. • **EN_IOTA_CFG_LOG_LOCAL_NUMBER**: used to set logs of the facility type. This parameter is valid only when syslog is used. • **EN_IOTA_CFG_LOG_LEVEL**: used to set the level of logs to display. This parameter is valid only when syslog is used. • **EN_IOTA_CFG_KEEP_ALIVE_TIME**: used to set the MQTT link keepalive period. The client sends an MQTT ping message to keep alive. If this parameter is not set, the default value 120s is used. • **EN_IOTA_CFG_CONNECT_TIMEOUT**: used to set the MQTT connection timeout period. If this parameter is not set, the default value 30s is used. • **EN_IOTA_CFG_RETRY_INTERVAL**: used to set the reconnection attempt interval. If this parameter is not set, the default value 10s is used. • **EN_IOTA_CFG_QOS**: used to set the QoS for message publishing. If this parameter is not set, the default value 1 is used. • **EN_IOTA_CFG_AUTH_MODE**: used to set the device access mode. The secret and certificate access modes are available.
pValue/uiValue (value)	Mandatory	HW_CHAR */HW_UINT	Value of each configuration item. • Set **EN_IOTA_CFG_DEVICEID** to the device ID returned during device registration. • Set **EN_IOTA_CFG_DEVICESECRET** to the device secret returned during device registration. • Set **EN_IOTA_CFG_MQTT_ADDR** to the IP address of the IoT platform to which the SDK is connected. • Set **EN_IOTA_CFG_MQTT_PORT** to

Parameter	Mandatory or Optional	Type	Description
			8883. • Set **EN_IOTA_CFG_LOG_LOCAL_NUMBER** to the log source, which can be any one from LOG_LOCAL0 to LOG_LOCAL7. • Set **EN_IOTA_CFG_LOG_LEVEL** to **LOG_ERR**, **LOG_WARNING**, **LOG_INFO**, or **LOG_DEBUG**. • Set **EN_IOTA_CFG_KEEP_ALIVE_TIME** as required. Its value is measured in seconds. • Set **EN_IOTA_CFG_CONNECT_TIMEOUT** as required. Its value is measured in seconds. • Set **EN_IOTA_CFG_RETRY_INTERVAL** as required. Its value is measured in seconds. • Set **EN_IOTA_CFG_QOS** to **0** (at most once), **1** (at least once), or **2** (exact once). If this parameter is not set, the default value 1 is used. • Set **EN_IOTA_CFG_AUTH_MODE** to **EN_IOTA_CFG_AUTH_MODE_SECRET** (secret access) or **EN_IOTA_CFG_AUTH_MODE_CERT** (certificate access).

Return Value

For details, see Function Return Values.

Example

```
// Call this API to configure parameters.
    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 Setting Log Printing Callback Functions

API Function

This API is used to customize callback functions to print SDK logs.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
pfnLogCallbackHandler	Mandatory	PFN_LOG_CALLBACK_HANDLER	Name of a custom function.

PFN_LOG_CALLBACK_HANDLER type:

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

Parameter	Mandatory or Optional	Type	Description
level	Mandatory	int	Log level. The options are as follows: • EN_LOG_LEVEL_DEBUG • EN_LOG_LEVEL_INFO • EN_LOG_LEVEL_WARNING • EN_LOG_LEVEL_ERROR
format	Mandatory	char*	Log content, which can contain variables to print.
args	Optional	char*/int	Variable values to print.

Return Value

For details, see Function Return Values.

Example

```
// Customize the log printing functions.
void myPrintLog(int level, char* format, va_list args)
{
   vprintf(format, args); // Logs are printed on the console.
//  vsyslog(level, format, args); // Logs are recorded in the system log file.
}

// Set the custom log printing functions.
IOTA_SetPrintLogCallback(myPrintLog);
```

NOTE

- Logs printed by the **vprintf** function are displayed on the console.
- Logs printed by the **vsyslog** function are recorded in the system log file. Generally, the logs are stored in the **/var/log/messages** file. (You can use several packages as required.) You are advised to implement the log printing functions.

  Debug logs in the Linux operating system are required only during debugging. To display debug logs on the console, lower the debug log level.

  For example:

  ```
  void myPrintLog(int level, char* format, va_list args)
  {
    switch (level)
    {
         case EN_LOG_LEVEL_DEBUG:
              vsyslog(LOG_WARNING, format, args);// Lower the debug log level before
  printing logs.
              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 Setting Callback Functions

You can set callback functions to enable devices to process received downstream data. The downstream data includes data at the protocol layer and service layer. The SDK automatically subscribes to topics related to the service layer. For details on the downstream data of the service layer, see the API parameter description.

3.5.1 Setting Callback Functions at the Protocol Layer

API Description

The **IOTA_SetProtocolCallback** function is used to set callback functions at the protocol layer.

```
HW_VOID IOTA_SetProtocolCallback(HW_INT iItem, PFN_PROTOCOL_CALLBACK_HANDLER
pfnCallbackHandler)
```

Parameters

Parameter	Mandatory or Optional	Type	Description
iItem	Mandatory	HW_INT	Notification corresponding to the callback function. 1. **EN_IOTA_CALLBACK_CONNECT_SUCCESS**: notification indicating that the device authentication is successful 2. **EN_IOTA_CALLBACK_CONNECT_FAILURE**: notification indicating that the device authentication fails 3. **EN_IOTA_CALLBACK_CONNECTION_LOST**: notification indicating that the device is disconnected from the platform 4. **EN_IOTA_CALLBACK_DISCONNECT_SUCCESS**: notification indicating that the device proactively disconnects from the platform 5. **EN_IOTA_CALLBACK_DISCONNECT_FAILURE**: notification indicating that the device fails to disconnect from the platform 6. **EN_IOTA_CALLBACK_SUBSCRIBE_SUCCESS**: notification indicating that the device successfully subscribes to the topic 7. **EN_IOTA_CALLBACK_SUBSCRIBE_FAILURE**: notification indicating that the device fails to subscribe to the topic 8. **EN_IOTA_CALLBACK_PUBLISH_SUCCESS**: notification indicating that the device successfully publishes data 9. **EN_IOTA_CALLBACK_PUBLISH_FAILURE**: notification indicating that the device fails to publish data

Parameter	Mandatory or Optional	Type	Description
pfnCallbackHandler	Mandatory	PFN_PROTOCOL_CALLBACK_HANDLER	Name of the custom function.

PFN_CALLBACK_HANDLER type:

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

EN_IOTA_MQTT_PROTOCOL_RSP type:

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO	Message returned by the MQTT protocol layer.
message	Mandatory	HW_CHAR*	Message body.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Example

```
// Set the callback functions.
// For details on the HandleConnectSuccess and HandleConnectFailure functions, see the
implementation of src/device_demo/device_demo.c in the SDK.
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 Setting Callback Functions at the Service Layer

API Description

The following callback functions are at the service layer.

API Function	Description	Input Parameter Description
HW_VOID IOTA_SetMessageCallback(PFN_MESSAGE_CALLBACK_HANDLER pfnCallbackHandler)	Used to set message callback functions.	Custom callback function pointer. For details, see 5.1 Receiving a Message.
HW_VOID IOTA_SetUserTopicMsgCallback(PFN_USER_TOPIC_MSG_CALLBACK_HANDLER pfnCallbackHandler)	Used to set the callback functions for messages to a custom topic.	Custom callback function pointer. For details, see 5.2.2 Receiving a Message from a Custom Topic.
HW_VOID IOTA_SetCmdCallback(PFN_CMD_CALLBACK_HANDLER pfnCallbackHandler)	Used to set command callback functions.	Custom callback function pointer. For details, see 5.3 Receiving a Command.
HW_VOID IOTA_SetPropSetCallback(PFN_PROP_SET_CALLBACK_HANDLER pfnCallbackHandler)	Used to set the callback functions for setting device properties.	Custom callback function pointer. For details, see 5.4 Receiving a Device Property Setting Request.

API Function	Description	Input Parameter Description
HW_VOID IOTA_SetPropGetCallback(PFN_PROP_GET_CALLBACK_HANDLER pfnCallbackHandler)	Used to set the callback functions for querying device properties.	Custom callback function pointer. For details, see 5.5 Receiving a Device Property Query Request.
HW_VOID IOTA_SetEventCallback(PFN_EVENT_CALLBACK_HANDLER pfnCallbackHandler)	Used to set event callback (**Events include child device addition or deletion and OTA upgrade**).	Custom callback function pointer. For details on the child device processing, see 5.6 Receiving a Child Device Addition Notification and 5.7 Receiving a Child Device Deletion Notification. For details on the OTA processing, see 7.1 Platform Requesting the Software or Firmware Version and 7.3 Platform Delivering an Upgrade Notification.
HW_VOID IOTA_SetShadowGetCallback(PFN_SHADOW_GET_CALLBACK_HANDLER pfnCallbackHandler)	Used to set the device shadow callback functions.	Custom callback function pointer. For details, see 8.2 Receiving Shadow Data.
HW_API_FUNC HW_VOID IOTA_SetDeviceConfigCallback(PFN_DEVICE_CONFIG_CALLBACK_HANDLER pfnCallbackHandler)	Used to set the callback function for the remote device configuration result.	Custom callback function pointer. For details, see 12.2 Receiving the Remote Configuration Result.
HW_API_FUNC HW_VOID IOTA_SetM2mCallback(PFN_M2M_CALLBACK_HANDLER pfnCallbackHandler)	Used to set the M2M callback function.	Custom callback function pointer.
HW_API_FUNC HW_VOID IOTA_SetDeviceRuleSendMsgCallback(PFN_DEVICE_RULE_SEND_MSG_CALLBACK_HANDLER pfnCallbackHandler)	Used to set the callback function for sending rule engine messages on the device.	Custom callback function pointer.

Example

```
// Set the callback functions.
// Functions such as HandleMessageDown and HandleUserTopicMessageDown are demo
functions in device_demo.
void SetMyCallbacks() {
       // Callback function at the protocol layer
    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);

       // Callback function at the service layer
    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 Connecting to the Platform

API Function

This API is used by a device to connect to the IoT platform.

API Description

```
HW_INT IOTA_Connect()
```

Return Value

For details, see Function Return Values.

Example

```
 // Connect to the platform.
int ret = IOTA_Connect();
if (ret != 0)
{
   printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: IOTA_Connect() error, Auth failed,
result %d\n", ret);
}
```

3.7 Disconnecting from the Platform

API Function

This API is used by a device to proactively disconnect from the IoT platform.

API Description

```
HW_INT IOTA_DisConnect()
```

Return Value

For details, see Function Return Values.

Example

```
// Disconnect from the platform.
int ret = IOTA_DisConnect();
if (ret != 0)
{
   printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: IOTA_DisConnect() error, DisAuth
failed, result %d\n", ret);
}
```


4 Data Reporting from Directly Connected Devices

4.1 Reporting a Message to a Custom Topic

API Function

This API is used by a directly connected device to report a message that will not be parsed by the platform.

API Description

```
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)
```

Parameters

Parameter	Mandatory or Optional	Type	Description
object_device_id	Optional	HW_CHAR*	Target device. If this parameter is set to **NULL**, the destination device is a gateway.

Parameter	Mandatory or Optional	Type	Description
name	Optional	HW_CHAR*	Message name.
id	Optional	HW_CHAR*	Unique identifier of the message.
content	Mandatory	HW_CHAR*	Message content.
topicParas	Optional	HW_CHAR*	Custom topic parameters, for example, **devMsg**. (Do not add '/' or special characters before the parameter). If this parameter is set to **NULL**, the default topic of the platform is used for data reporting.
compressFlag	Mandatory	HW_INT	Whether to compress data before data reporting. The value **0** means not to compress the data, and **1** means to compress the data. Compression consumes memory. If there is no specific traffic requirement, non-compressed reporting is recommended.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

Return Value

For details, see Function Return Values.

Example

```
// Report a message.
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 Reporting Properties

API Function

This API is used by a directly connected device to report properties that can be parsed by the IoT platform. The device properties must be predefined in the product model.

API Description

```
HW_INT IOTA_PropertiesReport(ST_IOTA_SERVICE_DATA_INFO pServiceData[], HW_INT
serviceNum, HW_INT compressFlag, void *context)
```

Parameters

Parameter	Mandatory or Optional	Type	Description
pServiceData[]	Mandatory	ST_IOTA_SERVICE_DATA_INFO	Structure of the properties to report.
serviceNum	Mandatory	HW_INT	Number of services to report.
compressFlag	Mandatory	HW_INT	Whether to compress data before data reporting. The value **0** means not to compress the data, and **1** means to compress the data. Compression consumes memory. If there is no specific traffic requirement, non-compressed reporting is recommended.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_SERVICE_DATA_INFO type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	HW_CHAR*	ID of a service, which can be obtained from the product model.
event_time	Optional	HW_CHAR*	UTC time when the device collects data. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**. If this parameter is set to **NULL**, the time when the platform receives the data reported is used as the data reporting time.
properties	Mandatory	HW_CHAR*	Data of the service. The fields are defined in the product model. Note: The format must be able to be parsed into JSON. For details, see the following example.

Return Value

For details, see Function Return Values.

Example

```
// Report device properties.
void Test_propertiesReport()
{
    int serviceNum = 2; // Number of services to be reported by the device.
    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 Reporting the Command Execution Result

API Function

This API is used by a directly connected device to report the command execution result after the device receives a command (5.3) delivered by the IoT platform.

API Description

```
HW_INT IOTA_CommandResponse(HW_CHAR *requestId, HW_INT result_code, HW_CHAR
*response_name, HW_CHAR *pcCommandResponse, void *context)
```

Parameters

Parameter	Mandatory or Optional	Type	Description

Parameter	Mandatory or Optional	Type	Description
requestId	Mandatory	HW_CHAR*	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.
result_code	Mandatory	HW_INT	Execution result. **0** indicates a successful execution, whereas other values indicate an execution failure. If this parameter is not carried, the execution is considered to be successful.
response_name	Optional	HW_CHAR*	Response name, which is defined in the product model.
pcCommandResponse	Mandatory	HW_CHAR*	Response result. The value must be the same as that of **commandResponse** defined in the product model. Note: The format must be able to be parsed into JSON. For details, see the following example.
context	Optional	void*	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

Return Value

For details, see Function Return Values.

Example

```
// Report the command execution result.
    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 Reporting the Property Setting Result

API Function

This API is used by a directly connected device to report the property setting result after the device receives a property setting command (5.4) from the IoT platform.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
requestId	Mandatory	HW_CHAR*	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.
result_code	Mandatory	HW_INT	Execution result. **0** indicates a successful execution, whereas other values indicate an execution failure. If this parameter is not carried, the execution is considered to be successful.
result_desc	Optional	HW_CHAR*	Description of the property setting response, which can be set to **NULL**.
context	Optional	void*	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

Return Value

For details, see Function Return Values.

Example

```
// Report the property setting result.
//  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 Reporting the Property Query Result

API Function

This API is used by a directly connected device to report the property query result after the device receives a property query command (5.5) from the IoT platform.

API Description

```
HW_INT IOTA_PropertiesGetResponse(HW_CHAR *requestId, ST_IOTA_SERVICE_DATA_INFO
serviceProp[], HW_INT serviceNum, void *context);
```

Parameters

Parameter	Mandatory or Optional	Type	Description
requestId	Mandatory	HW_CHAR *	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.
serviceProp	Optional	ST_IOTA_SERVICE_DATA_INFO	Structure of the device properties. Note: If this parameter is not carried, set **serviceNum** to **0**.
serviceNum	Mandatory	HW_INT	Number of services to report.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_SERVICE_DATA_INFO type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	HW_CHAR *	ID of a service, which can be obtained from the product model.
event_time	Optional	HW_CHAR *	UTC time when the device collects data. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**.

Parameter	Mandatory or Optional	Type	Description
			If this parameter is set to **NULL**, the platform time is used as the data reporting time.
properties	Mandatory	HW_CHAR *	Data of the service. The fields are defined in the product model. Note: The format must be able to be parsed into JSON. For details, see the following example.

Return Value

For details, see Function Return Values.

Example

```
// Report the property query result.
    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 Reporting the Child Device Status

API Function

This API is used by a gateway to update the child device status.

API Description

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


Parameters

Parameter	Mandatory or Optional	Type	Description
device_statuses	Mandatory	ST_IOTA_DEVICE_STATUSES*	List of child device statuses to report.
deviceNum	Mandatory	HW_INT	Number of child devices.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_SERVICE_DATA_INFO type:

Parameter	Mandatory or Optional	Type	Description
event_time	Optional	HW_CHAR *	Event time.
device_statuses[]	Mandatory	ST_IOTA_DEVICE_STATUS	Child device status list.

ST_IOTA_DEVICE_STATUS type:

Parameter	Mandatory or Optional	Type	Description
device_id	Mandatory	HW_CHAR *	Child device ID.
status	Mandatory	HW_CHAR *	Child device status. **OFFLINE**: The device is offline. **ONLINE**: The device is online.

Return Value

For details, see Function Return Values.

Example

```
// Report the child device status.
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 Requesting for Adding a Child Device

API Function

This API is used to add a child device to a gateway.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
subDevicesInfo	Mandatory	ST_IOTA_SUB_DEVICE_INFO *	Child device information structure.
deviceNum	Mandatory	HW_INT	Number of child devices to add at a time. Maximum value: 50. If there are more child devices, add them in groups.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_SUB_DEVICE_INFO type:

Parameter	Mandatory or Optional	Type	Description
event_time	Optional	HW_CHAR *	Event time. If this parameter is not carried, the platform uses the UTC time. The default value is **NULL**.
event_id	Optional	HW_CHAR *	Event ID, which is used to identify a request. If this parameter is not carried, the value is generated by the

Parameter	Mandatory or Optional	Type	Description
			platform by default. The default value is **NULL**.
deviceInfo	Mandatory	ST_IOTA_DEVICE_INFO	Information about the child device to add.

ST_IOTA_DEVICE_INFO type:

Parameter	Mandatory or Optional	Type	Description
parent_device_id	Optional	HW_CHAR *	Parent device ID.
node_id	Mandatory	HW_CHAR *	Node ID.
device_id	Optional	HW_CHAR *	Device ID.
name	Optional	HW_CHAR *	Device name.
description	Optional	HW_CHAR *	Device description.
product_id	Mandatory	HW_CHAR *	Product ID.
extension_info	Optional	HW_CHAR *	Extended device information, which can be customized. The maximum size of the value is 1 KB.

Return Value

For details, see Function Return Values.

For details about the response parameters of the platform, see 5.8 Receiving the Response to the Request for Adding a Child Device.

Example

```
// Add a child device for a directly connected device.
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 Requesting for Deleting a Child Device

API Function

This API is used to delete a child device to a gateway.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
delSubDevices	Mandatory	ST_IOTA_DEL_SUB_DEVICE*	Child device information structure.
deviceNum	Mandatory	HW_INT	Number of child devices to delete at a time. Maximum value: 50. If there are more child devices, delete them in groups.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_DEL_SUB_DEVICE type:

Parameter	Mandatory or Optional	Type	Description
event_time	Optional	HW_CHAR *	Event time. If this parameter is not carried, the platform uses the UTC time. The default value is **NULL**.
event_id	Optional	HW_CHAR *	Event ID, which is used to identify a request. If this parameter is not carried, the value is generated by the platform by default. The default value is **NULL**.
delSubDevice	Mandatory	HW_CHAR **	List of child device statuses to delete.

Return Value

For details, see Function Return Values.

For details about the response parameters of the platform, see 5.9 Receiving the Response to the Request for Deleting a Child Device.

Example

```
// Delete a child device for a directly connected device.
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 Data Receipt for Directly Connected Devices

Devices can receive commands from the IoT platform. (The SDK automatically subscribes to related topics.) The commands include device messages, commands, device property setting or query requests, and child device addition or deletion notifications.

You can set callback functions to enable the devices to process commands. For details, see 3.5.2 Setting Callback Functions at the Service Layer. This section describes the **message body** (input parameters of the callback functions) and the implementation of the callback functions.

NOTE

- The structure of the downstream functions is released by the SDK. To save the content in the structure, copy the content during service processing.
- The third party determines whether to deliver optional parameters in the message body.

5.1 Receiving a Message

API Description

A device receives a message delivered by an application. The IoT platform does not parse the message.

Message Body

EN_IOTA_MESSAGE structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	String	Target device corresponding to the message. If this parameter is set to **NULL**, the destination device is a gateway.
name	Optional	String	Message name.
id	Optional	String	Unique identifier of the message.
content	Mandatory	String	Message content.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Example

```
// Implement the processing on a message.
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);
}

// Set the callback function.
IOTA_SetMessageCallback(HandleMessageDown);
```

5.2 Topic Processing

A device can receive messages from a custom topic only after subscribing to the topic.

5.2.1 Subscribing to a Custom Topic

API Function

This API is used to subscribe to a custom topic.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
topicParas	Mandatory	HW_CHAR*	Custom topic parameters, for example, **devMsg**. Do not add '/' or special characters before the parameter.

Return Value

For details, see Function Return Values.

Example

```
// Subscribe to a custom topic.
IOTA_SubscribeUserTopic("devMsg");
```

5.2.2 Receiving a Message from a Custom Topic

API Description

A device receives a message from a custom topic delivered by an application. The IoT platform does not parse the message.

Message Body

EN_IOTA_USER_TOPIC_MESSAGE structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	String	Target device corresponding to the message. If this parameter is set to **NULL**, the destination device is a gateway. For example, if a gateway receives a message on behalf of a child device, the child device is the destination device.
name	Optional	String	Message name.
id	Optional	String	Unique identifier of the message.
content	Mandatory	String	Message content.
topic_para	Mandatory	String	Custom topic parameter, for example, **devMsg**.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Example

```
// Implement the processing on a message from a custom topic.
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);
}

// Set the callback function.
IOTA_SetUserTopicMsgCallback(HandleUserTopicMessageDown);
```

5.2.3 Subscribing to a Bootstrap Topic

API Function

This API is used to subscribe to bootstrap topics.

API Description

```
HW_API_FUNC HW_INT IOTA_SubscribeBoostrap()
```

Example

```
// Subscribe to a bootstrap topic.
IOTA_SubscribeBoostrap();
```

5.3 Receiving a Command

API Description

A device receives a command from the IoT platform.

Message Body

EN_IOTA_COMMAND structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	HW_CHAR*	ID of the device to which the platform delivers a command.
service_id	Optional	HW_CHAR*	Service ID of the device.
command_name	Optional	HW_CHAR*	Command name, which is defined in the product model.
paras	Mandatory	HW_CHAR*	Command execution parameters, which are defined in the product

Parameter	Mandatory or Optional	Type	Description
			model.
request_id	Mandatory	HW_CHAR*	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Example

```
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);
    }
}

// Implement the processing on a command.
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
}

// Set the callback function.
IOTA_SetCmdCallback(HandleCommandRequest);
```

NOTE

The **Test_commandResponse(requestId)** function implements the command response logic. For details, see 4.3 Reporting the Command Execution Result.

5.4 Receiving a Device Property Setting Request

Description

A device receives the properties set by the IoT platform.

Message Body

Table 5-1 EN_IOTA_PROPERTY_SET structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	HW_CHAR*	ID of the device whose properties are to be set.
request_id	Mandatory	HW_CHAR*	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.
services	Mandator	EN_IOTA_SER	Service list.

Parameter	Mandatory or Optional	Type	Description
	y	VICE_PROPERTY*	
services_count	Mandatory	HW_INT	Number of services.

Table 5-2 EN_IOTA_SERVICE_PROPERTY type

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	HW_CHAR*	Service ID of the device.
properties	Mandatory	HW_CHAR*	Service properties, which are defined in the product model.

Table 5-3 EN_IOTA_MQTT_MSG_INFO type

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Example

```
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);
    }
}
```

```
// Implement the processing on a request for setting device properties.
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
}


// Set the callback function.
IOTA_SetPropSetCallback(HandlePropertiesSet);
```

NOTE

The **Test_propSetResponse(requestId)** function implements the command response logic. For details, see 4.4 Reporting the Property Setting Result.

5.5 Receiving a Device Property Query Request

Description

A device receives a property query request from the IoT platform.

Message Body

Table 5-4 EN_IOTA_PROPERTY_GET structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_dev	Optional	HW_CHAR*	ID of the device whose properties are to be queried. If this parameter is set to **NULL**,

Parameter	Mandatory or Optional	Type	Description
ice_id			the properties of the gateway are queried.
service_id	Optional	HW_CHAR*	Service ID of the device. If this parameter is set to **NULL**, all services are queried.
request_id	Mandatory	HW_CHAR*	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.

Table 5-5 EN_IOTA_MQTT_MSG_INFO type

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Example

```
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);
}

// Implement the processing on a request for querying device properties.
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
}

// Set the callback function.
IOTA_SetPropGetCallback(HandlePropertiesGet);
```

NOTE

The **Test_propGetResponse(requestId)** function implements the command response logic. For details, see 4.5 Reporting the Property Query Result.

5.6 Receiving a Child Device Addition Notification

Description

The IoT platform notifies a gateway that a child device is added.

Message Body

Table 5-6 EN_IOTA_EVENT structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.

Parameter	Mandatory or Optional	Type	Description
object_device_id	Optional	HW_CHAR*	Target device that the event is about. If this parameter is set to **NULL**, the destination device is a gateway.
services	Optional	EN_IOTA_SERVICE_EVENT*	List of services that the event is about.
services_count	Optional	int	Number of services.

Table 5-7 EN_IOTA_MQTT_MSG_INFO type

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Table 5-8 EN_IOTA_SERVICE_EVENT type

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	int	EN_IOTA_EVENT_SUB_DEVICE_MANAGER (enumerated value: **0**)
event_type	Mandatory	int	EN_IOTA_EVENT_ADD_SUB_DEVICE_NOTIFY (enumerated value: **0**)
event_time	Optional	HW_CHAR*	Event time.
paras	Mandatory	EN_IOTA_DEVICE_PARAS*	Child device event parameters.
ota_paras	Optional	EN_IOTA_OTA_PARAS*	OTA event parameters. In this case, the value is **NULL**.
ntp_paras	Optional	EN_IOTA_NTP	NTP event parameters. In this case, the

Parameter	Mandatory or Optional	Type	Description
		_PARAS*	value is **NULL**.
gtw_add_device_paras	Optional	EN_IOTA_GTW_ADD_DEVICE_PARAS	Event of adding a child device to a gateway. In this case, the value is **NULL**.
gtw_del_device_paras	Optional	EN_IOTA_GTW_DEL_DEVICE_PARAS	Event of deleting a child device for a gateway. In this case, the value is **NULL**.

Table 5-9 EN_IOTA_DEVICE_PARAS structure

Parameter	Mandatory or Optional	Type	Description
devices	Mandatory	EN_IOTA_DEVICE_INFO*	Device list.
devices_count	Mandatory	int	Number of devices.
version	Mandatory	long long	Child device version.

Table 5-10 EN_IOTA_DEVICE_INFO structure

Parameter	Mandatory or Optional	Type	Description
parent_device_id	Mandatory	HW_CHAR*	Parent device ID.
node_id	Mandatory	HW_CHAR*	Node ID.
device_id	Mandatory	HW_CHAR*	Device ID.
name	Optional	HW_CHAR*	Device name.
description	Optional	HW_CHAR*	Device description.
manufacturer_id	Optional	HW_CHAR*	Manufacturer ID.
model	Optional	HW_CHAR*	Device model.
product_id	Optional	HW_CHAR*	Product ID.
fw_version	Optional	HW_CHAR*	Firmware version.
sw_version	Optional	HW_CHAR*	Software version.
status	Optional	HW_CHAR*	Device status.

Parameter	Mandatory or Optional	Type	Description
			ONLINE: The device is online. **OFFLINE**: The device is offline.
extension_info	Optional	HW_CHAR*	Extended device information, which can be customized.

Example

```
// Set the callback function.
IOTA_SetCallback(EN_IOTA_CALLBACK_EVENT_DOWN, HandleEventsDown);

// Implement event-related command processing.
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--;
        }
}
```

NOTE

- **if (message->services[i].event_type == EN_IOTA_EVENT_ADD_SUB_DEVICE_NOTIFY)** determines that the event is a child device addition notification and reports a data record on behalf of the child device.
- **if (message->services[i].event_type == EN_IOTA_EVENT_DELETE_SUB_DEVICE_NOTIFY)** determines that the event is a child device deletion notification.
- **if (message->services[i].event_type == EN_IOTA_EVENT_ADD_SUB_DEVICE_RESPONSE)** determines that the event is a response to a child device addition.
- **if (message->services[i].event_type == EN_IOTA_EVENT_DEL_SUB_DEVICE_RESPONSE)** determines that the event is a response to a child device deletion.
- The **Test_UpdateSubDeviceStatus** function is used to report the child device status. For details, see 4.6 Reporting the Child Device Status.
- The **Test_BatchPropertiesReport** function is used to report child device data. For details, see 6.2 Reporting Device Properties in Batches.

5.7 Receiving a Child Device Deletion Notification

Description

The IoT platform notifies a gateway that a child device is deleted.

Message Body

Table 5-11 EN_IOTA_EVENT structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	String	Target device that the event is about. If this parameter is set to **NULL**, the destination device is a gateway.
services	Optional	EN_IOTA_SERVICE_EVENT*	List of services that the event is about.
services_count	Optional	int	Number of services.

Table 5-12 EN_IOTA_MQTT_MSG_INFO type

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Table 5-13 EN_IOTA_SERVICE_EVENT type

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	int	EN_IOTA_EVENT_SUB_DEVICE_MANAGER (enumerated value: **0**)
event_type	Mandatory	int	EN_IOTA_EVENT_DELETE_SUB_DEVICE_NOTIFY (enumerated value: **1**)
event_time	Optional	String	Event time.

Parameter	Mandatory or Optional	Type	Description
paras	Mandatory	EN_IOTA_DEVICE_PARAS*	Child device event parameters.
ota_paras	Optional	EN_IOTA_OTA_PARAS*	OTA event parameters. In this case, the value is **NULL**.
ntp_paras	Optional	EN_IOTA_NTP_PARAS*	NTP event parameters. In this case, the value is **NULL**.
gtw_add_device_paras	Optional	EN_IOTA_GTW_ADD_DEVICE_PARAS	Event of adding a child device to a gateway. In this case, the value is **NULL**.
gtw_del_device_paras	Optional	EN_IOTA_GTW_DEL_DEVICE_PARAS	Event of deleting a child device for a gateway. In this case, the value is **NULL**.

Table 5-14 EN_IOTA_DEVICE_PARAS structure

Parameter	Mandatory or Optional	Type	Description
devices	Mandatory	EN_IOTA_DEVICE_INFO*	Device list.
devices_count	Mandatory	int	Number of devices.
version	Mandatory	long long	Child device version.

Table 5-15 EN_IOTA_DEVICE_PARAS structure

Parameter	Mandatory or Optional	Type	Description
devices	Mandatory	EN_IOTA_DEVICE_INFO*	Device list.
devices_count	Mandatory	int	Number of devices.
version	Mandatory	long long	Child device version.

Table 5-16 EN_IOTA_DEVICE_INFO structure

Parameter	Mandatory or Optional	Type	Description
parent_device_id	Mandatory	String	Parent device ID.
node_id	Optional	String	Node ID.
device_id	Mandatory	String	Device ID.

Example

```
For details, see 5.6 Receiving a Child Device Addition Notification.
```

NOTE

The topics used for child device addition and deletion events are the same. The **event_type** field in the message body can be used to determine whether the topic is used for child device addition or deletion.

5.8 Receiving the Response to the Request for Adding a Child Device

Description

The IoT platform sends a response to the request for adding a child device.

Message Body

EN_IOTA_EVENT structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	String	Target device that the event is about. If this parameter is set to **NULL**, the destination device is a gateway.
services	Optional	EN_IOTA_SERVICE_EVENT*	List of services that the event is about.
services_count	Optional	int	Number of services.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

EN_IOTA_SERVICE_EVENT type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	int	EN_IOTA_EVENT_SUB_DEVICE_MANAGER (enumerated value: **0**)
event_type	Mandatory	int	EN_IOTA_EVENT_ADD_SUB_DEVICE_RESPONSE (Enumerated value: **6**)
event_time	Optional	String	Event time.
paras	Mandatory	EN_IOTA_DEVICE_PARAS*	Child device event parameters. In this case, the value is **NULL**.
ota_paras	Optional	EN_IOTA_OTA_PARAS*	OTA event parameters. In this case, the value is **NULL**.
ntp_paras	Optional	EN_IOTA_NTP_PARAS*	NTP event parameters. In this case, the value is **NULL**.
gtw_add_device_paras	Mandatory	EN_IOTA_GTW_ADD_DEVICE_PARAS	Event of adding a child device to a gateway.
gtw_del_device_paras	Optional	EN_IOTA_GTW_DEL_DEVICE_PARAS	Event of deleting a child device for a gateway. In this case, the value is **NULL**.
device_log_paras	Optional	EN_IOTA_DEVICE_LOG_PARAS*	Device log event parameter. In this case, the value is **NULL**.

EN_IOTA_GTW_ADD_DEVICE_PARAS structure:

Parameter	Mandatory or Optional	Type	Description
successful_devices	Mandatory	EN_IOTA_DEVICE_INFO*	Details about the child devices that are added.
successful_devices_count	Mandatory	int	Number of child devices that are added.
failed_devices	Mandatory	EN_IOTA_ADD_DEVICE_FAILED_REASON*	Cause for the failure to add child devices.
failed_devices_count	Mandatory	int	Number of child devices that fail to be added.

EN_IOTA_DEVICE_INFO structure:

Parameter	Mandatory or Optional	Type	Description
parent_device_id	Mandatory	String	Parent device ID.
node_id	Mandatory	String	Node ID.
device_id	Mandatory	String	Device ID.
name	Optional	String	Device name.
description	Optional	String	Device description.
manufacturer_id	Optional	String	Manufacturer ID.
model	Optional	String	Device model.
product_id	Optional	String	Product ID.
fw_version	Optional	String	Firmware version.
sw_version	Optional	String	Software version.
status	Optional	String	Device status. **ONLINE**: The device is online. **OFFLINE**: The device is offline.
extension_info	Optional	Object	Extended device information, which can be customized.

EN_IOTA_ADD_DEVICE_FAILED_REASON structure:

Parameter	Mandatory or Optional	Type	Description
node_id	Mandatory	String	The value is that of **node_id** of the corresponding device in the corresponding request.
product_id	Mandatory	String	The value is that of **product_id** of the corresponding device in the corresponding request.
error_code	Mandatory	String	Error code.
error_msg	Mandatory	String	Error description.

Example

```
For details, see 5.6 Receiving a Child Device Addition Notification.
```

NOTE

The topic of the notification about adding or deleting a child device on the IoT platform is the same as the topic of the response to the request for adding or deleting a child device on the gateway. You can determine the topic based on the **event_type** field in the message body.

5.9 Receiving the Response to the Request for Deleting a Child Device

Description

The IoT platform sends a response to the request for deleting a child device.

Message Body

EN_IOTA_EVENT structure:

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	String	Target device of the event. If this parameter is set to **NULL**, the destination device is a gateway.
services	Optional	EN_IOTA_SERVICE_EVENT*	List of services that an event is about.
services_coun	Optional	int	Number of services.

Parameter	Mandatory or Optional	Type	Description
t			

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

EN_IOTA_SERVICE_EVENT type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	int	EN_IOTA_EVENT_SUB_DEVICE_MANAGER (enumerated value: **0**)
event_type	Mandatory	int	EN_IOTA_EVENT_DEL_SUB_DEVICE_RESPONSE (enumerated value: **7**)
event_time	Optional	String	Event time.
paras	Mandatory	EN_IOTA_DEVICE_PARAS*	Child device event parameters. In this case, the value is **NULL**.
ota_paras	Optional	EN_IOTA_OTA_PARAS*	OTA event parameters. In this case, the value is **NULL**.
ntp_paras	Optional	EN_IOTA_NTP_PARAS*	NTP event parameters. In this case, the value is **NULL**.
gtw_add_device_paras	Mandatory	EN_IOTA_GTW_ADD_DEVICE_PARAS	Event of adding a child device to a gateway. In this case, the value is **NULL**.
gtw_del_device_paras	Optional	EN_IOTA_GTW_DEL_DEVICE_PARAS	Event of deleting a child device to a gateway.

Parameter	Mandatory or Optional	Type	Description
device_log_paras	Optional	EN_IOTA_DEVICE_LOG_PARAS*	Device log event parameter. In this case, the value is **NULL**.

EN_IOTA_GTW_ADD_DEVICE_PARAS structure:

Parameter	Mandatory or Optional	Type	Description
successful_devices	Mandatory	EN_IOTA_DEVICE_INFO *	Details about the child devices that are deleted.
successful_devices_count	Mandatory	int	Number of child devices that are deleted.
failed_devices	Mandatory	EN_IOTA_DEL_DEVICE_FAILED_REASON *	Cause for the failure to delete child devices.
failed_devices_count	Mandatory	int	Number of child devices that fail to be deleted.

EN_IOTA_DEL_DEVICE_FAILED_REASON structure:

Parameter	Mandatory or Optional	Type	Description
device_id	Mandatory	String	The value is that of **device_id** of the corresponding device in the corresponding request.
error_code	Mandatory	String	Error code.
error_msg	Mandatory	String	Error description.

EN_IOTA_DEVICE_INFO structure:

Parameter	Mandatory or Optional	Type	Description
parent_device_id	Mandatory	String	Parent device ID.

Parameter	Mandatory or Optional	Type	Description
node_id	Mandatory	String	Node ID.
device_id	Mandatory	String	Identifier of a device.

Example

```
For details, see 5.6 Receiving a Child Device Addition Notification.
```

NOTE

The topic of the notification about adding or deleting a child device on the IoT platform is the same as the topic of the response to the request for adding or deleting a child device on the gateway. You can determine the topic based on the **event_type** field in the message body.

6 Data Reporting from Child Devices

6.1 Reporting a Message

To enable a child device to report a message to the platform through a gateway, set **object_device_id** to the ID of the child device. For details, see 4.1 Reporting a Message to a Custom Topic.

6.2 Reporting Device Properties in Batches

API Function

This API is used by a gateway to report data of multiple child devices to the IoT platform.

API Description

```
HW_INT IOTA_BatchPropertiesReport(ST_IOTA_DEVICE_DATA_INFO pDeviceData[], HW_INT
deviceNum, HW_INT serviceLenList[], HW_INT compressFlag, void *context)
```

Parameters

Parameter	Mandatory or Optional	Type	Description
pServiceData[]	Mandatory	ST_IOTA_DEVICE_DATA_INFO	Array hosting the structure of device data to report.
deviceNum	Mandatory	HW_INT	Number of child devices.
serviceLenList[]	Mandatory	HW_INT	Array hosting the number of services reported by child devices.
compressFlag	Mandatory	HW_INT	Whether to compress data before data

Parameter	Mandatory or Optional	Type	Description
			reporting. The value **0** means not to compress the data, and **1** means to compress the data. Compression consumes memory. If there is no specific traffic requirement, non-compressed reporting is recommended.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_SERVICE_DATA_INFO type:

Parameter	Mandatory or Optional	Type	Description
device_id	Mandatory	HW_CHAR *	Device ID.
services[MaxServiceReportNum]	Mandatory	ST_IOTA_SERVICE_DATA_INFO	Data of the service. The fields are defined in the product model. Note: The format must be able to be parsed into JSON. For details, see the following example. **MaxServiceReportNum** indicates the maximum number of services contained in a device data record. The default value is **10**, which can be customized.

ST_IOTA_SERVICE_DATA_INFO type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	HW_CHAR *	ID of a service, which can be obtained from the product model.
event_time	Optional	HW_CHAR *	UTC time when the device collects data. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**. If this parameter is set to **NULL**, the platform time is used as the data reporting time.
properties	Mandatory	HW_CHAR *	Data of the service. The fields are defined in the product model. Note: The format must be able to be parsed into JSON. For

Parameter	Mandatory or Optional	Type	Description
			details, see the following example.

Example

```
// Report device properties in batches.
    int deviceNum = 1;     // Number of child devices to report data
    ST_IOTA_DEVICE_DATA_INFO devices[deviceNum]; // Array hosting the structure of data
to be reported by child devices
    int serviceList[deviceNum]; // Number of services to be reported by each child device
   serviceList[0] = 2;      //  device1 needs to report two services.
//  serviceList[1] = 1;          // device2 needs to report one service.


   char *device1_service1 = "{\"mno\":\"1\",\"imsi\":\"3\"}"; // Properties to be
reported by service1 (in JSON format)

   char *device1_service2 =
"{\"hostCpuUsage\":\"2\",\"containerCpuUsage\":\"4\"}";// Properties to be reported
by service2 (in JSON format)

    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 Upgrade

7.1 Platform Requesting the Software or Firmware Version

Description

The IoT platform requests to obtain version information.

Message Body

EN_IOTA_EVENT structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	HW_CHAR*	Target device that the event is about. If this parameter is set to **NULL**, the destination device is a gateway.
services	Optional	EN_IOTA_SERVICE_EVENT*	List of services that the event is about.
services_count	Optional	int	Number of services.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

EN_IOTA_SERVICE_EVENT type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	int	EN_IOTA_EVENT_OTA (enumerated value: **1**)
event_type	Mandatory	int	EN_IOTA_EVENT_VERSION_QUERY (enumerated value: **2**)
event_time	Optional	String	Event time.
paras	Mandatory	EN_IOTA_DEVICE_PARAS*	Child device event parameters. In this case, the value is **NULL**.
ota_paras	Optional	EN_IOTA_OTA_PARAS*	OTA event parameters.
ntp_paras	Optional	EN_IOTA_NTP_PARAS*	NTP event parameters. In this case, the value is **NULL**.
gtw_add_device_paras	Optional	EN_IOTA_GTW_ADD_DEVICE_PARAS	Event of adding a child device to a gateway. In this case, the value is **NULL**.
gtw_del_device_paras	Optional	EN_IOTA_GTW_DEL_DEVICE_PARAS	Event of deleting a child device for a gateway. In this case, the value is **NULL**.
device_log_paras	Optional	EN_IOTA_DEVICE_LOG_PARAS*	Device log event parameter. In this case, the value is **NULL**.

ota_paras structure

Parameter	Mandatory or Optional	Type	Description
-	-	-	-

Example

```
For details, see 5.6 Receiving a Child Device Addition Notification.
```

7.2 Device Reporting the Software or Firmware Version

API Function

This API is used by a device to report the software or firmware version.

API Description

```
HW_INT IOTA_OTAVersionReport(ST_IOTA_OTA_VERSION_INFO otaVersionInfo, void *context)
```

Parameters

Parameter	Mandatory or Optional	Type	Description
otaVersionInfo	Mandatory	ST_IOTA_OTA_VERSION_INFO	Software/Firmware version data structure.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_OTA_VERSION_INFO type:

Parameter	Mandatory or Optional	Type	Description
sw_version	Mandatory	HW_CHAR*	Software version. Either **sw_version** or **fw_version** must not be **NULL**.
fw_version	Mandatory	HW_CHAR*	Firmware version. Either **sw_version** or **fw_version** must not be **NULL**.
event_time	Optional	HW_CHAR*	UTC time when the device collects data. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**.

Parameter	Mandatory or Optional	Type	Description
			If this parameter is set to **NULL**, the platform time is used as the data reporting time.
object_device_id	Optional	HW_CHAR*	Target device that the event is about. If this parameter is set to **NULL**, the device specified in the topic is considered to be the device involved.

Return Value

For details, see Function Return Values.

Example

```
// Report the software or firmware version.
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 Platform Delivering an Upgrade Notification

Description

The IoT platform delivers an upgrade notification.

Message Body

EN_IOTA_EVENT structure

Parameter	Mandatory or Optional	Type	Description

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	String	Target device that the event is about. If this parameter is set to **NULL**, the destination device is a gateway.
services	Optional	EN_IOTA_SERVICE_EVENT*	List of services that the event is about.
services_count	Optional	int	Number of services.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

EN_IOTA_SERVICE_EVENT type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	int	EN_IOTA_EVENT_OTA (enumerated value: **1**)
event_type	Mandatory	int	• EN_IOTA_EVENT_FIRMWARE_UPGRADE (enumerated value: **3**, indicating firmware upgrade) • EN_IOTA_EVENT_SOFTWARE_UPGRADE (enumerated value: **4**, indicating software upgrade)
event_time	Optional	String	Event time.

Parameter	Mandatory or Optional	Type	Description
paras	Mandatory	EN_IOTA_DEVICE_PARAS*	Child device event parameters. In this case, the value is **NULL**.
ota_paras	Optional	EN_IOTA_OTA_PARAS*	OTA event parameters.
ntp_paras	Optional	EN_IOTA_NTP_PARAS*	NTP event parameters. In this case, the value is **NULL**.
gtw_add_device_paras	Optional	EN_IOTA_GTW_ADD_DEVICE_PARAS	Event of adding a child device to a gateway. In this case, the value is **NULL**.
gtw_del_device_paras	Optional	EN_IOTA_GTW_DEL_DEVICE_PARAS	Event of deleting a child device for a gateway. In this case, the value is **NULL**.
device_log_paras	Optional	EN_IOTA_DEVICE_LOG_PARAS*	Device log event parameter. In this case, the value is **NULL**.

ota_paras structure

Parameter	Mandatory or Optional	Type	Description
version	Mandatory	String	Software or firmware package version.
url	Mandatory	String	Address for downloading the software or firmware package.
file_size	Mandatory	Integer	Software or firmware package size.
access_token	Optional	String	Temporary token of the URL for downloading the software or firmware package.
expires	Optional	Integer	Time when the **access_token** expires.
sign	Mandatory	String	MD5 value for the software or firmware package.

Example

```
For details, see 5.6 Receiving a Child Device Addition Notification.
```


7.4 Device Requesting a Software or Firmware Package

7.4.1 Downloading an OTA Package into the Current Directory

API Function

This API is used by a gateway to apply for downloading a software or firmware package in the current directory.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
url	Mandatory	HW_CHAR*	Address for downloading the software or firmware package.
token	Mandatory	HW_CHAR*	Temporary token of the URL for downloading the software or firmware package.
serviceNum	Mandatory	timeout	Timeout interval for downloading a package. The value must be greater than 300 seconds. It is recommended that the value is less than 24 hours.

Return Value

For details, see Function Return Values.

Example

```
For details, see 5.6 Receiving a Child Device Addition Notification.
```

7.4.2 Downloading an OTA Package into a Specific Path

API Function

This API is used by a gateway to apply for downloading a software or firmware package into the specified directory and obtain the file name.

API Description

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


Parameters

Parameter	Mandatory or Optional	Type	Description
url	Mandatory	HW_CHAR*	Address for downloading the software or firmware package.
token	Mandatory	HW_CHAR*	Temporary token of the URL for downloading the software or firmware package.
serviceNum	Mandatory	timeout	Timeout interval for downloading a package. The value must be greater than 300 seconds. It is recommended that the value is less than 24 hours.
otaFilePath	Mandatory	HW_CHAR*	Software/Firmware package storage path.
otaFilenameOut	Mandatory	HW_CHAR*	Software/Firmware package name.

Return Value

For details, see Function Return Values.

Example

```
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 Device Reporting the Upgrade Status

API Function

This API is used by a device to report the upgrade status.

API Description

```
HW_INT IOTA_OTAStatusReport(ST_IOTA_UPGRADE_STATUS_INFO otaStatusInfo, void *context)
```

Parameters

Parameter	Mandatory or Optional	Type	Description
otaStatusInfo	Mandatory	ST_IOTA_UPGRADE_STATUS_INFO	Upgrade status data structure.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_UPGRADE_STATUS_INFO type:

Parameter	Mandatory or Optional	Type	Description
result_code	Mandatory	HW_INT	Device upgrade status. **0**: successful upgrade **1**: device in use **2**: poor signal **3**: already the latest version **4**: low battery **5**: insufficient space **6**: download timeout **7**: update package verification failure **8**: unsupported upgrade package type **9**: insufficient memory **10**: upgrade package installation failure **255**: internal exception
progress	Optional	HW_INT	Device upgrade progress. The value ranges from 0 to 100.
event_time	Optional	HW_CHAR*	UTC time when the device collects data. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**. If this parameter is set to **NULL**, the platform time is used as the data reporting time.
object_devic	Optional	HW_CHAR*	Target device that the event is about. If

Parameter	Mandatory or Optional	Type	Description
e_id			this parameter is set to **NULL**, the device specified in the topic is considered to be the device involved.
description	Optional	HW_CHAR*	Description of the upgrade status, such as the cause for upgrade failure.
version	Mandatory	HW_CHAR*	Current version of the device.

Return Value

For details, see Function Return Values.

Example

```
// Report the upgrade status.
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 Device Shadow

8.1 Requesting Shadow Data

API Function

This API is used by a device to request shadow data from the IoT platform.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
requestId	Mandatory	HW_CHAR*	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.
service_id	Optional	HW_CHAR*	Service of the device that requests the device shadow. If this parameter is set to **NULL**, device shadow data of all services will be returned.
object_device_id	Optional	HW_CHAR*	Target device that the event is about. If this parameter is set to **NULL**, the device specified in the topic is considered to be the device involved.
context	Optional	void *	Context pointer that points to any data

Parameter	Mandatory or Optional	Type	Description
			body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

Return Value

For details, see Function Return Values.

Example

```
// Request shadow data.
    //get device shadow
    IOTA_GetDeviceShadow("1232", NULL, NULL, NULL);
```

8.2 Receiving Shadow Data

API Description

A device receives shadow data from the IoT platform.

Message Body

EN_IOTA_EVENT structure

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	HW_CHAR*	ID of the device that requests its device shadow.
request_id	Mandatory	HW_CHAR*	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.
shadow	Optional	EN_IOTA_SHADOW_DATA*	Shadow data list.
shadow_data_count	Optional	int	Number of service shadows. The default value is **0**.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

EN_IOTA_SHADOW_DATA type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	String	Service ID of the device.
desired_properties	Optional	String	Properties in the desired section of the device shadow, which are defined in the product model associated with the device.
desired_event_time	Optional	String	UTC time of the device property data in the desired section of the device shadow. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**.
reported_properties	Optional	String	Properties in the reported section of the device shadow, which are defined in the product model associated with the device.
reported_event_time	Optional	String	UTC time of the device property data in the reported section of the device shadow. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**.
version	Optional	Integer	Device shadow version.

Example

```
// Implement shadow data processing.
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++;
    }
}

// Set the callback function.
IOTA_SetShadowGetCallback(HandleDeviceShadowRsp);
```


9 Device Time Synchronization

9.1 Requesting for Time Synchronization

API Function

This API is used by a device to send a request for synchronizing time with the platform.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

Return Value

For details, see Function Return Values.

Example

```
// Send a time synchronization request.
//NTP
IOTA_GetNTPTime(NULL);
```


9.2 Receiving a Response for Time Synchronization

Description

The device receives the time synchronization response delivered by the IoT platform and obtains the accurate device time.

Message Body

EN_IOTA_EVENT structure:

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	HW_CHAR*	Target device that the event is about. If this parameter is set to **NULL**, the destination device is a gateway.
services	Optional	EN_IOTA_SERVICE_EVENT*	List of services that the event is about.
services_count	Optional	int	Number of services.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

EN_IOTA_SERVICE_EVENT type:

Parameter	Mandatory or Optional	Type	Description

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	int	EN_IOTA_EVENT_TIME_SYNC (enumerated value: **2**)
event_type	Mandatory	int	EN_IOTA_EVENT_GET_TIME_SYNC_RESPONSE (enumerated value: **5**)
event_time	Optional	HW_CHAR*	Event time.
paras	Mandatory	EN_IOTA_DEVICE_PARAS*	Child device event parameters. In this case, the value is **NULL**.
ota_paras	Optional	EN_IOTA_OTA_PARAS*	OTA event parameters. In this case, the value is **NULL**.
ntp_paras	Mandatory	EN_IOTA_NTP_PARAS*	NTP event parameters.
gtw_add_device_paras	Optional	EN_IOTA_GTW_ADD_DEVICE_PARAS	Event of adding a child device to a gateway. In this case, the value is **NULL**.
gtw_del_device_paras	Optional	EN_IOTA_GTW_DEL_DEVICE_PARAS	Event of deleting a child device for a gateway. In this case, the value is **NULL**.
device_log_paras	Optional	EN_IOTA_DEVICE_LOG_PARAS*	Device log event parameter. In this case, the value is **NULL**.

EN_IOTA_NTP_PARAS structure:

Parameter	Mandatory or Optional	Type	Description
device_real_time	Mandatory	HW_LLONG	Actual device time.

Example

```
For details, see 5.6 Receiving a Child Device Addition Notification.
```


10 Device Reporting Information

10.1 Reporting Information

API Function

This API is used by a device to report device information to the IoT platform. After the device is connected to the IoT platform, the device automatically reports a message carrying the SDK version number.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
device_info_report	Optional	ST_IOTA_DEVICE_INFO_REPORT *	Device information structure.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_DEVICE_INFO_REPORT type:

Parameter	Mandatory or Optional	Type	Description
object_device_id	Optional	String	Device that an event is about. If this parameter is not carried, the device specified in the topic is considered to be the device involved.
event_time	Optional	String	Event time.
device_sdk_version	Optional	String	The value is in the format of *access mode_version number*, for example, **C_v0.5.0**, **JAVA_v0.5.0**, or **Tiny SDK_v1.0.0**.
sw_version	Optional	String	Software version.
fw_version	Optional	String	Firmware version.

Return Value

For details, see Function Return Values.

Example

```
// Report device information.
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 Device Reporting Logs

11.1 Platform Delivering a Log Collection Notification

Description

This API is used by the platform to deliver a log collection notification to devices.

Message Body

EN_IOTA_EVENT structure:

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
object_device_id	Optional	HW_CHAR*	Target device that the event is about. If this parameter is set to **NULL**, the destination device is a gateway.
services	Optional	EN_IOTA_SERVICE_EVENT*	List of services that the event is about.
services_count	Optional	HW_INT	Number of services.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

EN_IOTA_SERVICE_EVENT type:

Parameter	Mandatory or Optional	Type	Description
service_id	Mandatory	HW_INT	EN_IOTA_EVENT_DEVICE_LOG (enumerated value: **3**)
event_type	Mandatory	HW_INT	EN_IOTA_EVENT_LOG_CONFIG (enumerated value: **8**)
event_time	Optional	HW_CHAR*	Event time.
paras	Optional	EN_IOTA_DEVICE_PARAS*	Child device event parameters. In this case, the value is **NULL**.
ota_paras	Optional	EN_IOTA_OTA_PARAS*	OTA event parameters. In this case, the value is **NULL**.
ntp_paras	Optional	EN_IOTA_NTP_PARAS*	NTP event parameters. In this case, the value is **NULL**.
gtw_add_device_paras	Optional	EN_IOTA_GTW_ADD_DEVICE_PARAS*	Event of adding a child device to a gateway. In this case, the value is **NULL**.
gtw_del_device_paras	Optional	EN_IOTA_GTW_DEL_DEVICE_PARAS*	Event of deleting a child device for a gateway. In this case, the value is **NULL**.
device_log_paras	Mandatory	EN_IOTA_DEVICE_LOG_PARAS*	Device log event parameters.

EN_IOTA_DEVICE_LOG_PARAS structure:

Parameter	Mandatory or Optional	Type	Description
log_switch	Optional	HW_CHAR*	Device log collection switch. **on**: enables device log collection. **off**: disables device log collection.
end_time	Optional	HW_CHAR*	Time when log collection ends. yyyy-MM-dd'T'HH:mm:ss'Z'

Example

```
For details, see 5.6 Receiving a Child Device Addition Notification.
```

11.2 Reporting Logs to the Platform

API Function

When log collection is enabled, a device uses this API to report logs to the IoT platform. After the device is connected to the platform, the SDK automatically reports a login success log to the platform. The maximum size of the log content is 1 MB.

API Description

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

Parameter	Mandatory or Optional	Type	Description
type	Mandatory	HW_CHAR*	Log type: **DEVICE_STATUS**: device status. **DEVICE_PROPERTY**: device property. **DEVICE_MESSAGE**: device message. **DEVICE_COMMAND**: device command.
content	Mandatory	HW_CHAR*	Log content.
timestamp	Optional	HW_CHAR*	Timestamp when a log is generated, precise to millisecond.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a

Parameter	Mandatory or Optional	Type	Description
			successful or failed callback. If this pointer is not used, **NULL** can be passed.

Return Value

For details, see Function Return Values.

Example

```
// Send a time synchronization request.
//report device log
long long timestamp = getTime();
char timeStampStr[14];
sprintf(timeStampStr,"%lld",timestamp);
IOTA_ReportDeviceLog("DEVICE_STATUS", "device log", timeStampStr, NULL);
```


12 Remote SSH Login

12.1 Reporting the Remote SSH Configuration Result

API Function

This API is used to report the remote SSH configuration result of a device.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
device_config_report	Mandatory	const ST_IOTA_DEVICE_CONFIG_RESULT *	Structure of the remote device configuration result.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

ST_IOTA_DEVICE_CONFIG_RESULT structure:

Parameter	Mandatory or Optional	Type	Description
object_devic	Optional	HW_CHAR*	Target device. If this parameter is set to

Parameter	Mandatory or Optional	Type	Description
e_id			**NULL**, the destination device is a gateway.
result_code	Mandatory	HW_INT	Device configuration result. The value **0** indicates success, and other values indicate failure. If this parameter is not carried, the execution is considered successful.
description	Optional	HW_CHAR[]	Report result.

Return Value

For details, see Function Return Values.

Example

```
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;
}
The OnEventsDownRemoteCfgArrived function is invoked in the OnEventsDownArrived
function.
```

12.2 Receiving the Remote Configuration Result

API Description

This API is used to receive the remote device configuration result returned by the IoT platform. You can implement the **HandleDeviceConfig** function as required.

Example

```
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;
}
// Set the callback function.
IOTA_SetDeviceConfigCallback(HandleDeviceConfig);
```


13 Device-side Rules

13.1 Storing Device-side Rules Locally

API Function

This API is used to store device-side rules locally to cope with device outage scenarios.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
filepath	Mandatory	const char *	File path for storing rules.

Example

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


13.2 Calling Back the Function for Cross-Device Message Sending

API Description

This API is used to call back the function for cross-device message sending in the rule engine. You can implement the **HandleDeviceRuleSendMsg** function for message communication.

Example

```
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;
}
// Set the callback function.
IOTA_SetDeviceRuleSendMsgCallback(HandleDeviceRuleSendMsg);
```


14 IoTEdge M2M Communication

14.1 Sending M2M Messages

API Function

This API is used to send messages through M2M communication.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
to	Mandatory	HW_CHAR*	Target device ID.
from	Mandatory	HW_CHAR*	Source device ID.
content	Mandatory	HW_CHAR*	Content of the message to be sent.
requestId	Optional	HW_CHAR*	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

Return Value

For details, see Function Return Values.

Example

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

14.2 Receiving M2M Messages

API Description

This API is used to receive messages through M2M communication.

Message Body

EN_IOTA_M2M_MESSAGE structure:

Parameter	Mandatory or Optional	Type	Description
mqtt_msg_info	Mandatory	EN_IOTA_MQTT_MSG_INFO*	MQTT protocol layer information.
request_id	Mandatory	HW_CHAR *	Unique identifier of a request. If this parameter is carried in a message received by a device, the device must include the parameter value in the response sent to the platform.
to	Mandatory	HW_CHAR *	Target device ID.
from	Mandatory	HW_CHAR *	Source device ID.
content	Mandatory	HW_CHAR *	Message content.

EN_IOTA_MQTT_MSG_INFO type:

Parameter	Mandatory or Optional	Type	Description
context	Mandatory	HW_VOID*	A pointer to any application-specific context. The context pointer is passed to each callback function to provide access to the context in the callback. This parameter

Parameter	Mandatory or Optional	Type	Description
			is reserved. The default value is **NULL**.
messageId	Mandatory	HW_INT	Message ID.
code	Mandatory	HW_INT	Message code. (The default value is **0** for a service layer notification.)

Example

```
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
}
// Set the callback function.
IOTA_SetM2mCallback(HandleM2mMessageDown);
```


15 Device-Cloud Secure Communication

15.1 Requesting for the Latest Soft Bus Information

API Function

This API is used to proactively obtain the latest soft bus information configured on the IoT platform.

API Description

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

Parameters

Parameter	Mandatory or Optional	Type	Description
busId	Optional	HW_CHAR*	Unique ID of the soft bus generated by the cloud platform. Entered by the user. If the value is **NULL**, the latest information about all soft buses of the current device is queried. If the value is not **NULL**, the latest information about a specific soft bus is queried.
eventId	Optional	HW_CHAR*	Event ID, which is used to identify a request.
context	Optional	void *	Context pointer that points to any data body. The pointer is returned in a successful or failed callback. If this pointer is not used, **NULL** can be passed.

Return Value

For details, see Function Return Values.

Example

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

