IoTHub client “C” Library

Revision 1.6

09/21/2015

# Revisions:

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Updated By | Date | Major updates |
| 0.1 | TA | 01/15/2015 | Initial draft |
| 0.2 | TA | 01/22/2015 | Reviewed comments and simplified API |
| 0.3 | TA | 01/30/2015 | Completed API for both the IOTHUB\_message & IOTHUB\_client |
| 1.0 | TA | 03/05/2015 | Updated based on the final implementation of the IOTHUB\_client APIs |
| 1.1 | TE | 04/14/2015 | Adjust the config to use a function pointer rather than an enum |
| 1.2 | DC | 04/19/2015 | Rename DeviceHub to IoTHub |
| 1.3 | AP | 05/18/2015 | Rework to bring \_DoWork and threadlessness. |
| 1.5 | AP | 07/02/2015 | Add \_SetOption |
| 1.6 | TE | 9/24/2015 | Rework to bring up to date with current API |

# Overview

The IoTHub client “C” library offers developers a means of communication to & from an IoT Hub.

The library allows a device to achieve the supported communication patterns by a IoT Hub:

* **Event**. A message sent by a device to a IoT Hub. The device does not expect a response from IoT Hub, but the event message can result in errors (e.g. not being processed …).
* **Message**. A message sent by IoT Hub to the device. IoT Hub does not expect a response from the device, but the message can result in errors (e.g. failed delivery…)

The library offers the following features:

* The library provides an encapsulation for the message transfer between a device & IoT Hub.
* The library supports AMQP, HTTP, and MQTT protocols for communication with a IoT Hub.
* The library uses Proton-C to establish the AMQP communication.
* The library uses Paho to establish the MQTT communication.
* The library is a static lib.
* The source code for the library will be available under the “IOTHUB\_client” under the “Azure/azure-iot-sdks” in the GitHub repository.
* The repository is located at: <https://github.com/Azure/azure-iot-sdks>

The APIs of this library cannot be called from different threads on the same handle without risking data races. Therefore, should more than 1 thread need to access concurrently the APIs of this module on the same handle, there needs to be a user-level synchronization mechanism that guarantees that two APIs are not called at the same time.

# Example1 – SendEventAsync:

#include <stdio.h>

#include <stdlib.h>

#include "iothub\_client.h"

#include "iothub\_message.h"

#include "threadapi.h"

#include "crt\_abstractions.h"

#include "iothubtransportamqp.h"

#define MAX\_NUMBER\_OF\_MESSAGES 5

static const unsigned int SLEEP\_IN\_MILLISECONDS = 500;

static const int MESSAGE\_BASE\_TRACKING\_ID = 42;

static int g\_callbackInvoked;

typedef struct EVENT\_INSTANCE\_TAG

{

IOTHUB\_MESSAGE\_HANDLE messageHandle;

int messageTrackingId; // For tracking the messages within the user callback.

} EVENT\_INSTANCE;

static void ReceiveConfirmationCallback(IOTHUB\_CLIENT\_CONFIRMATION\_RESULT result, void\* userContextCallback)

{

EVENT\_INSTANCE\* eventInstance = (EVENT\_INSTANCE\*)userContextCallback;

(void)printf("Confirmation[%d] received for message tracking id = %d with result = %d\r\n", g\_callbackInvoked, eventInstance->messageTrackingId, result);

/\* Some device specific action code goes here... \*/

g\_callbackInvoked++;

}

int main(void)

{

const char\* connectionString = "[device connection string]";

IOTHUB\_CLIENT\_HANDLE iotHubClientHandle;

EVENT\_INSTANCE eventInstance[MAX\_NUMBER\_OF\_MESSAGES];

size\_t msgLength = 1024;

const char\* msgText = malloc(msgLength);

g\_callbackInvoked = 0;

(void)printf("Starting the IoTHub client sample to Send Event Asynchronously...\r\n");

if ((iotHubClientHandle = IoTHubClient\_CreateFromConnectionString(connectionString, AMQP\_Protocol)) == NULL)

{

(void)printf("ERROR: iotHubClientHandle is NULL!\r\n");

}

else

{

IOTHUB\_CLIENT\_STATUS sendStatus;

for (int i = 0; i < MAX\_NUMBER\_OF\_MESSAGES; i++)

{

sprintf\_s((char\*)msgText, msgLength, "Message\_%d\_From\_IoTHubClient", i);

if ((eventInstance[i].messageHandle = IoTHubMessage\_CreateFromByteArray((const unsigned char\*)msgText, strlen(msgText))) == NULL)

{

(void)printf("ERROR: IoTHubMessageHandle is NULL!\r\n");

}

else

{

eventInstance[i].messageTrackingId = MESSAGE\_BASE\_TRACKING\_ID + i;

if (IoTHubClient\_SendEventAsync(iotHubClientHandle, eventInstance[i].messageHandle, ReceiveConfirmationCallback, &eventInstance[i]) != IOTHUB\_CLIENT\_OK)

{

(void)printf("ERROR: IoTHubClient\_SendEventAsync..........FAILED!\r\n");

}

else

{

(void)printf("IoTHubClient\_SendEventAsync accepted data for transmission to IoT Hub.\r\n");

}

(void)printf("IoTHubClient\_GetSendStatus() returns %d\r\n", IoTHubClient\_GetSendStatus(iotHubClientHandle,&sendStatus));

(void)printf("IoTHubClient\_GetSendStatus indicated a client status of: %d\r\n", sendStatus);

}

}

while (g\_callbackInvoked < MAX\_NUMBER\_OF\_MESSAGES)

{

ThreadAPI\_Sleep(SLEEP\_IN\_MILLISECONDS);

}

(void)printf("IoTHubClient\_GetSendStatus() returns %d\r\n", IoTHubClient\_GetSendStatus(iotHubClientHandle, &sendStatus));

(void)printf("IoTHubClient\_GetSendStatus indicated a client status of: %d\r\n", sendStatus);

}

IoTHubClient\_Destroy(iotHubClientHandle);

return 0;

}

# Example2 – ReceiveMessage

#include <stdio.h>

#include <stdlib.h>

#include "iothub\_client.h"

#include "iothub\_message.h"

#include "threadapi.h"

#include "crt\_abstractions.h"

#include "iothubtransportamqp.h"

#define MAX\_NUMBER\_OF\_MESSAGES 100

static const unsigned int SLEEP\_IN\_MILLISECONDS = 500;

static const int MESSAGE\_BASE\_TRACKING\_ID = 42;

static int g\_callbackInvoked;

static IOTHUBMESSAGE\_DISPOSITION\_RESULT ReceiveMessageCallback(IOTHUB\_MESSAGE\_HANDLE message, void\* userContextCallback)

{

    int\* callbackInvoked = (int\*)userContextCallback;

    const unsigned char\* payload;

size\_t size;

if (IoTHubMessage\_GetByteArray(message,&payload,&size) == IOTHUB\_MESSAGE\_OK)

{

    (void)printf("Received message [%d] with Data: <<<%.\*s>>> & Size=%d\r\n", \*callbackInvoked, (int)size, payload, (int)size);

     /\* Some device specific action code goes here... \*/

}

else

{

(void)printf("Received message [%d] with bad data\r\n",\*callbackInvoked);

}

(\*callbackInvoked)++;

    return IOTHUBMESSAGE\_ACCEPTED;

}

int main(void)

{

const char\* connectionString = "[device connection string]";

    IOTHUB\_CLIENT\_HANDLE iotHubClientHandle;

    g\_callbackInvoked = 0;

    (void)printf("Starting the IoTHub client sample to Receive messages...\r\n");

    if ((iotHubClientHandle = IoTHubClient\_CreateFromConnectionString(connectionString, AMQP\_Protocol)) == NULL)

    {

        (void)printf("ERROR: iotHubClientHandle is NULL!\r\n");

    }

    else

    {

        if (IoTHubClient\_SetMessageCallback(iotHubClientHandle, ReceiveMessageCallback, &g\_callbackInvoked) != IOTHUB\_CLIENT\_OK)

        {

            (void)printf("ERROR: IoTHubClient\_SetMessageCallback..........FAILED!\r\n");

        }

        else

        {

time\_t theTime;

            (void)printf("IoTHubClient\_SetMessageCallback...successful.\r\n");

while (g\_callbackInvoked < MAX\_NUMBER\_OF\_MESSAGES)

     {

         ThreadAPI\_Sleep(SLEEP\_IN\_MILLISECONDS);

     }

if(IoTHubClient\_GetLastMessageReceiveTime(iotHubClientHandle, &theTime)!=IOTHUB\_CLIENT\_OK)

{

printf("unable to provide the last received message time...\r\n");

}

else

{

printf("the time when the last message was received was %s\r\n", ctime(&theTime));

}

        }

    }

    IoTHubClient\_Destroy(iotHubClientHandle);

return 0;

}

# Types defined by the IoTHub client:

#define IOTHUB\_CLIENT\_RESULT\_VALUES       \

    IOTHUB\_CLIENT\_OK,                     \

    IOTHUB\_CLIENT\_INVALID\_ARG,            \

    IOTHUB\_CLIENT\_ERROR,                  \

    IOTHUB\_CLIENT\_INVALID\_SIZE,           \

    IOTHUB\_CLIENT\_INDEFINITE\_TIME         \

DEFINE\_ENUM(IOTHUB\_CLIENT\_RESULT, IOTHUB\_CLIENT\_RESULT\_VALUES);

#define IOTHUB\_CLIENT\_CONFIRMATION\_RESULT\_VALUES     \

    IOTHUB\_CLIENT\_CONFIRMATION\_OK,                   \

    IOTHUB\_CLIENT\_CONFIRMATION\_BECAUSE\_DESTROY,      \

    IOTHUB\_CLIENT\_CONFIRMATION\_ERROR                 \

DEFINE\_ENUM(IOTHUB\_CLIENT\_CONFIRMATION\_RESULT, IOTHUB\_CLIENT\_CONFIRMATION\_RESULT\_VALUES);

#define IOTHUB\_CLIENT\_STATUS\_VALUES       \

    IOTHUB\_CLIENT\_SEND\_STATUS\_IDLE,       \

    IOTHUB\_CLIENT\_SEND\_STATUS\_BUSY        \

DEFINE\_ENUM(IOTHUB\_CLIENT\_STATUS, IOTHUB\_CLIENT\_STATUS\_VALUES);

#define IOTHUBMESSAGE\_DISPOSITION\_RESULT\_VALUES \

    IOTHUBMESSAGE\_ACCEPTED, \

    IOTHUBMESSAGE\_REJECTED, \

    IOTHUBMESSAGE\_ABANDONED

DEFINE\_ENUM(IOTHUBMESSAGE\_DISPOSITION\_RESULT, IOTHUBMESSAGE\_DISPOSITION\_RESULT\_VALUES);

typedef void(\*IOTHUB\_CLIENT\_EVENT\_CONFIRMATION\_CALLBACK)(IOTHUB\_CLIENT\_CONFIRMATION\_RESULT result, void\* userContextCallback);

typedef IOTHUBMESSAGE\_DISPOSITION\_RESULT (\*IOTHUB\_CLIENT\_MESSAGE\_CALLBACK\_ASYNC)(IOTHUB\_MESSAGE\_HANDLE message, void\* userContextCallback);

typedef const void\*(\*IOTHUB\_CLIENT\_TRANSPORT\_PROVIDER)(void);

typedef struct IOTHUB\_CLIENT\_CONFIG\_TAG

{

    IOTHUB\_CLIENT\_TRANSPORT\_PROVIDER protocol;

    const char\* deviceId;

    const char\* deviceKey;

    const char\* iotHubName;

    const char\* iotHubSuffix;

    const char\* protocolGatewayHostName;

} IOTHUB\_CLIENT\_CONFIG;

# Types defined by the IoTHub message module:

#define IOTHUB\_MESSAGE\_RESULT\_VALUES \

IOTHUB\_MESSAGE\_OK, \

IOTHUB\_MESSAGE\_INVALID\_ARG, \

IOTHUB\_MESSAGE\_INVALID\_TYPE, \

IOTHUB\_MESSAGE\_ERROR \

DEFINE\_ENUM(IOTHUB\_MESSAGE\_RESULT, IOTHUB\_MESSAGE\_RESULT\_VALUES);

#define IOTHUBMESSAGE\_CONTENT\_TYPE\_VALUES \

IOTHUBMESSAGE\_BYTEARRAY, \

IOTHUBMESSAGE\_STRING, \

IOTHUBMESSAGE\_UNKNOWN \

DEFINE\_ENUM(IOTHUBMESSAGE\_CONTENT\_TYPE, IOTHUBMESSAGE\_CONTENT\_TYPE\_VALUES);

# IoTHub client Structures

## IOTHUB\_CLIENT\_CONFIG

### struct members

|  |  |
| --- | --- |
| Name | Description |
| protocol | A function pointer that is passed into the IoTHubClientCreate. A function definition for amqp, AMQP\_Protocol, is defined in the include iothubtransportamqp.h. A function definition for http, HTTP\_Protocol, is defined in the include iothubtransporthttp.h. A function definition for mqtt, MQTT\_Protocol, is defined in the include iothubtransportmqtt.h. |
| deviceId | A string that identifies the device. |
| deviceKey | The device key used to authenticate the device. |
| iotHubName | The IoT Hub name to which the device is connecting. |
| iotHubSuffix | The IoT Hub suffix goes here, e.g., private.azure-devices-int.net. |
| protocolGatewayHostName | The address of the protocol gateway the client will use to connect to send data and receive messages. This is for protocols which are supported via a protocol gateway. For example, mqtt. |

# IoTHub client APIs

## IOTHUB\_CLIENT\_HANDLE IoTHubClient\_CreateFromConnectionString(const char\* connectionString, IOTHUB\_CLIENT\_TRANSPORT\_PROVIDER protocol);

Creates a IoT Hub client for communication with an existing IoT Hub using the specified connection string parameter. The API does not allow sharing of a connection across multiple devices. This is a blocking call.

Sample connection string:

HostName=[IoT Hub name goes here].[IoT Hub suffix goes here, e.g., private.azure-devices-int.net]; DeviceId= [Device ID goes here];SharedAccessKey= [Device key goes here];

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| connectionString | Pointer to a character string |
| protocol | Function pointer for protocol implementation |

### Return

* A Non-NULL handle value that is used when invoking other functions for IoT Hub client.
* NULL on failure.

## IOTHUB\_CLIENT\_HANDLE IoTHubClient\_Create(const IOTHUB\_CLIENT\_CONFIG\* config);

Creates a IoT Hub client for communication with an existing IoT Hub using the specified parameters. The API does not allow sharing of a connection across multiple devices. This is a blocking call.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| config | Pointer to a IOTHUB\_CLIENT\_CONFIG structure |

### Return

* A Non-NULL handle value that is used when invoking other functions for IoT Hub client.
* NULL on failure.

## void IoTHubClient\_Destroy(IOTHUB\_CLIENT\_HANDLE iotHubClientHandle);

Disposes of resources allocated by the IoT Hub client. Any pending events that have not yet been sent to the IoT Hub will be immediately completed with a IOTHUB\_CLIENT\_CONFIRMATION\_BECAUSE\_DESTROY status. Other events that are actually out on the wire but not finished may receive an IOTHUB\_CLIENT\_CONFIRMATION\_ERROR status. This is a blocking call.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubClientHandle | The handle created by a call to the create function. |

### Return

* No return.

## IOTHUB\_CLIENT\_RESULT IoTHubClient\_SendEventAsync(IOTHUB\_CLIENT\_HANDLE iotHubClientHandle, IOTHUB\_MESSAGE\_HANDLE eventMessageHandle, IOTHUB\_CLIENT\_EVENT\_CONFIRMATION\_CALLBACK eventConfirmationCallback, void\* userContextCallback);

Asynchronous call to send the message specified by the IoTHubMessageHandle.

**NOTE: The application behavior is undefined if the user calls the IoTHubClient\_Destroy from within any callback.**

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubClientHandle | The handle created by a call to the create function. |
| eventMessageHandle | The handle to a IoT Hub message. |
| eventConfirmationCallBack | The callback specified by the device for receiving confirmation of the delivery of the IoT Hub message. This callback can be expected to invoke SendEventAsync for the same message in attempt to retry sending a failing message. The user can specify a NULL value here to indicate no callback required. |
| userContextCallback | User specified context that will be provided to the callback. This can be NULL. |

### Return

* IOTHUB\_CLIENT\_OK upon success.
* Error code upon failure.

## IOTHUB\_CLIENT\_RESULT IoTHubClient\_SetMessageCallback(IOTHUB\_CLIENT\_HANDLE iotHubClientHandle, IOTHUB\_CLIENT\_MESSAGE\_CALLBACK\_ASYNC messageCallback, void\* userContextCallback);

Sets up the message callback invoked when IoT Hub issues a message to the device. This is a blocking call. **NOTE: The application behavior is undefined if the user calls the IoTHubClient\_Destroy from within any callback.**

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubClientHandle | The handle created by a call to the create function. |
| messageCallback | The callback specified by the device for receiving messages from IoT Hub. |
| userContextCallback | User specified context that will be provided to the callback. This can be NULL. |

### Return

* IOTHUB\_CLIENT\_OK upon success.
* Error code upon failure.

## IOTHUB\_CLIENT\_RESULT IoTHubClient\_GetLastMessageReceiveTime (IOTHUB\_CLIENT\_HANDLE iotHubClientHandle, time\_t\* lastMessageReceiveTime);

This function returns in the out parameter lastMessageReceiveTime what was the value of the time() function when the last notification was received at the client.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubClientHandle | The handle created by a call to the create function. |
| lastMessageReceiveTime | Out parameter containing the value of time() function when the last message was received |

### Return

* IOTHUB\_CLIENT\_OK upon success.
* Error code upon failure.

## IOTHUB\_CLIENT\_RESULT IoTHubClient\_GetSendStatus(IOTHUB\_CLIENT\_HANDLE iotHubClientHandle, IOTHUB\_CLIENT\_STATUS \*iotHubClientStatus);

This function returns the current sending status for IoTHubClient**.**

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubClientHandle | The handle created by a call to the create function. |
| iotHubClientStatus | A pointer to an IOTHUB\_CLIENT\_STATUS. If the function call is successful then what is pointed to will receive: IOTHUBCLIENT\_SENDSTATUS\_IDLE if there are currently no items to be sent. IOTHUBCLIENT\_SENDSTATUS\_BUSY if there are currently items to be sent. |

### Return

* IOTHUB\_CLIENT\_OK upon success
* IOTHUBCLIENT \_INVALID\_ARG if called with NULL parameter
* Error code upon failure

## IOTHUB\_CLIENT\_MESSAGE\_CALLBACK\_ASYNC

Once a message is received from the service, if the user has set a callback, the receive callback shall be invoked.

This call back is defined as:

typedef IOTHUBMESSAGE\_DISPOSITION\_RESULT

(\*IOTHUB\_CLIENT\_MESSAGE\_CALLBACK\_ASYNC)(IOTHUB\_MESSAGE\_HANDLE message, void\* userContextCallback);

If the callback returns the status IOTHUBMESSAGE\_ACCEPTED, the client will accept the message, meaning that it will not be received again by the client. If the callback returns the status IOTHUBMESSAGE\_REJECTED , the message will be rejected. The message will not be resent to the device. If the callback returns the status IOTHUBMESSAGE\_ABANDONED, the message will be abandoned. The implies that the user could not process the message but it expected that the message will be resent to the device from the service. message is only valid in the scope of the callback.

## IOTHUB\_CLIENT\_RESULT IoTHubClient\_SetOption(IOTHUB\_CLIENT\_HANDLE iotHubClientHandle, const char\* optionName, const void\* value);

IoTHubClient\_SetOption shall set a runtime option identified by parameter "optionName" to a value pointed to by parameter value. The optionName and the data type value is pointing to are specific for every option.

Options:

"timeout" - the maximum time in miliseconds a communication is allowed to use. value is a pointer to an unsigned int with the meaning of "miliseconds". This is only supported for HTTP protocol so far. When the HTTP protocol uses CURL, the meaning of the parameter is total request time. When the HTTP protocol uses winhttp, the meaning is dwSendTimeout and dwReceiveTimeout parameters of [WinHttpSetTimeouts](https://msdn.microsoft.com/en-us/library/windows/desktop/aa384116(v=vs.85).aspx) API.

"[CURLOPT\_LOW\_SPEED\_LIMIT](http://curl.haxx.se/libcurl/c/CURLOPT_LOW_SPEED_LIMIT.html)" - only available for HTTP protocol and only when CURL is used. It has the same meaning as CURL's option with the same name. value is pointer to a long.

"[CURLOPT\_LOW\_SPEED\_TIME](http://curl.haxx.se/libcurl/c/CURLOPT_LOW_SPEED_TIME.html)" - only available for HTTP protocol and only when CURL is used. It has the same meaning as CURL's option with the same name. value is pointer to a long.

"[CURLOPT\_FORBID\_REUSE](http://curl.haxx.se/libcurl/c/CURLOPT_FORBID_REUSE.html)" - only available for HTTP protocol and only when CURL is used. It has the same meaning as CURL's option with the same name. value is pointer to a long.

"[CURLOPT\_FRESH\_CONNECT](http://curl.haxx.se/libcurl/c/CURLOPT_FRESH_CONNECT.html)" - only available for HTTP protocol and only when CURL is used. It has the same meaning as CURL's option with the same name. value is pointer to a long.

"[CURLOPT\_VERBOSE](http://curl.haxx.se/libcurl/c/CURLOPT_VERBOSE.html)" - only available for HTTP protocol and only when CURL is used. It has the same meaning as CURL's option with the same name. value is pointer to a long.

"messageTimeout" - maximum transmission time for an event from the moment when \_SendAsync is called. value is a pointer to a uint64\_t that contains the number of miliseconds after which events timeout. When the event times out, the message callback is invoked and the code IOTHUB\_CLIENT\_CONFIRMATION\_MESSAGE\_TIMEOUT is passed as parameter IOTHUB\_CLIENT\_CONFIRMATION\_RESULT result

By default messages never expire. The meaning of the messageTimeout value is the following:

0 = disable message timeout for all messages send by \_SendAsync from now on

Any other number - consider that number as the timeout.

## IotHubClient\_LL\_... APIs

IoTHubClient API surface also contains a separate set of APIs that allow the user to interact with the lower layer portion of the IoTHubClient. These APIs contain \_LL\_ in their name, but retain the same functionality like the IoTHubClient\_... APIs, with one difference. If the \_LL\_ APIs are used the user is responsible for scheduling when the actual work done by the IoTHubClient happens (when the data is sent/received on/from the wire). This is useful for constrained devices where spinning a separate thread is often not desired.

## IOTHUB\_CLIENT\_RESULT void IoTHubClient\_LL\_DoWork(IOTHUB\_CLIENT\_HANDLE iotHubClientHandle);

This function is user called when work (sending/receiving) can be done by the IoTHubClient**.** All IoTHubClient interactions (in regards to network trafic and/or user level callbacks) are the effect of calling this function and they take place synchronously inside \_DoWork.

Arguments:

|  |  |
| --- | --- |
| Name | Description |
| IotHubClientHandle | The handle created by a call to the create function. |

# Example of using DoWork:

#include <stdio.h>

#include <stdlib.h>

#include "iothub\_client.h"

#include "iothub\_message.h"

#include "threadapi.h"

#include "crt\_abstractions.h"

#include "iothubtransportamqp.h"

#define MAX\_NUMBER\_OF\_MESSAGES 5

static const unsigned int SLEEP\_IN\_MILLISECONDS = 500;

static const int MESSAGE\_BASE\_TRACKING\_ID = 42;

static int g\_callbackInvoked;

typedef struct EVENT\_INSTANCE\_TAG

{

IOTHUB\_MESSAGE\_HANDLE messageHandle;

int messageTrackingId; // For tracking the messages within the user callback.

} EVENT\_INSTANCE;

static void MessageConfirmationCallback(IOTHUB\_CLIENT\_CONFIRMATION\_RESULT result, void\* userContextCallback)

{

EVENT\_INSTANCE\* eventInstance = (EVENT\_INSTANCE\*)userContextCallback;

(void)printf("Confirmation[%d] received for message tracking id = %d with result = %d\r\n", g\_callbackInvoked, eventInstance->messageTrackingId, result);

/\* Some device specific action code goes here... \*/

g\_callbackInvoked++;

}

int main(void)

{

const char\* connectionString = "[device connection string]";

IOTHUB\_CLIENT\_HANDLE iotHubClientHandle;

EVENT\_INSTANCE eventInstance[MAX\_NUMBER\_OF\_MESSAGES];

size\_t msgLength = 1024;

const char\* msgText = malloc(msgLength);

g\_callbackInvoked = 0;

(void)printf("Starting the IoTHub client sample to Send Event Asynchronously...\r\n");

if ((iotHubClientHandle = IoTHubClient\_LL\_CreateFromConnectionString(connectionString, AMQP\_Protocol)) == NULL)

{

(void)printf("ERROR: iotHubClientHandle is NULL!\r\n");

}

else

{

IOTHUB\_CLIENT\_RESULT sendResult,getResult;

IOTHUB\_CLIENT\_STATUS sendStatus;

for (int i = 0; i < MAX\_NUMBER\_OF\_MESSAGES; i++)

{

sprintf\_s((char\*)msgText, msgLength, "Message\_%d\_From\_IoTHubClient", i);

if ((eventInstance[i].messageHandle = IoTHubMessage\_CreateFromString(msgText)) == NULL)

{

(void)printf("ERROR: IoTHubMessageHandle is NULL!\r\n");

}

else

{

eventInstance[i].messageTrackingId = MESSAGE\_BASE\_TRACKING\_ID + i;

if ((sendResult = IoTHubClient\_LL\_SendEventAsync(iotHubClientHandle, eventInstance[i].messageHandle, MessageConfirmationCallback, &eventInstance[i])) != IOTHUB\_CLIENT\_OK)

{

(void)printf("ERROR: IoTHubClient\_LL\_SendEventAsync..........FAILED! Status is: %d\r\n",sendResult);

}

else

{

(void)printf("IoTHubClient\_LL\_SendEventAsync accepted data for transmission to IoT Hub.\r\n");

}

getResult = IoTHubClient\_LL\_GetSendStatus(iotHubClientHandle, &sendStatus);

if (getResult != IOTHUB\_CLIENT\_OK)

{

(void)printf("IoTHubClient\_LL\_GetSendStatus failed! status is: %d", getResult);

}

else

{

(void)printf("IoTHubClient\_LL\_GetSendStatus() returns %d\r\n", sendStatus);

}

}

}

while (g\_callbackInvoked < MAX\_NUMBER\_OF\_MESSAGES)

{

IoTHubClient\_LL\_DoWork(iotHubClientHandle);

getResult = IoTHubClient\_LL\_GetSendStatus(iotHubClientHandle, &sendStatus);

if (getResult != IOTHUB\_CLIENT\_OK)

{

(void)printf("IoTHubClient\_LL\_GetSendStatus failed! status is: %d", getResult);

}

else

{

(void)printf("IoTHubClient\_LL\_GetSendStatus() returns %d\r\n", sendStatus);

}

ThreadAPI\_Sleep(SLEEP\_IN\_MILLISECONDS);

}

}

IoTHubClient\_LL\_Destroy(iotHubClientHandle);

return 0;

}

# IoTHub message APIs

## IOTHUB\_MESSAGE\_HANDLE IoTHubMessage\_CreateFromByteArray(const unsigned char\* byteArray, size\_t size) ;

Creates an IoT Hub message to be used for operations between the device and IoT Hub. The data pointed to by byteArray may contain unprintable data and is not zero terminated.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| byteArray | A pointer to data for the message. |
| size | The number of unsigned chars pointed to by byteArray. If size is zero then byteArray may be NULL. If size is not zero then byteArray MUST NOT be NULL. |

### Return

* A None-NULL handle value that is used when invoking other functions for IoT Hub message.
* NULL on failure.

## IOTHUB\_MESSAGE\_HANDLE IoTHubMessage\_CreateFromString(const char\* source);

Creates an IoT Hub message to be used for operations between the device and IoT Hub. The data is assumed to be printable and is zero terminated.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| source | A pointer to data for the message. |

### Return

* A None-NULL handle value that is used when invoking other functions for IoT Hub message.
* NULL on failure.

## IOTHUB\_MESSAGE\_HANDLE IoTHubMessage\_Clone(IOTHUB\_MESSAGE\_HANDLE iotHubMessageHandle);

Creates a new IoT Hub message with the content identical to that of the iotHubMessageHandle parameter.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubMessageHandle | Handle to the message to be cloned. |

### Return

* A None-NULL handle value that is used when invoking other functions for IoT Hub message.
* NULL on failure.

## IOTHUB\_MESSAGE\_RESULT IoTHubMessage\_GetByteArray(IOTHUB\_MESSAGE\_HANDLE iotHubMessageHandle, const unsigned char\*\* buffer, size\_t\* size);

Fetches a pointer and size for the data associated with the IoT Hub message handle. If the content type of the message is not IOTHUBMESSAGE\_BYTEARRAY then the function returns IOTHUB\_MESSAGE\_INVALID\_ARG.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubMessageHandle | Handle to the message to be cloned. |
| buffer | Pointer to the memory location where the pointer to the buffer will be written. |
| size | The size of the buffer will be written to this address. |

### Return

* IOTHUB\_MESSAGE\_OK if the byte array was fetched successfully.
* Error code upon failure.

## const char\* IoTHubMessage\_GetString(IOTHUB\_MESSAGE\_HANDLE iotHubMessageHandle);

Returns the zero terminated string stored in message. If the content type of the message is not IOTHUBMESSAGE\_STRING then the function returns NULL.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubMessageHandle | Handle to the message. |

### Return

* Pointer to the zero terminated string stored in the message.
* NULL if an error occurs or the content type is incorrect.

## IOTHUBMESSAGE\_CONTENT\_TYPE IoTHubMessage\_GetContentType(IOTHUB\_MESSAGE\_HANDLE iotHubMessageHandle);

Returns the content type of the message given by the parameter iotHubMessageHandle.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubMessageHandle | Handle to the message. |

### Return

* IOTHUBMESSAGE\_BYTEARRAY for a message created via IoTHubMessage\_CreateFromByteArray
* IOTHUBMESSAGE\_STRING for a message created via IoTHubMessage\_CreateFromString
* IOTHUBMESSAGE\_UNKNOWN otherwise.

## MAP\_HANDLE IoTHubMessage\_Properties(IOTHUB\_MESSAGE\_HANDLE iotHubMessageHandle);

Returns a handle to the message’s properties map.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| iotHubMessageHandle | Handle to the message. |

### Return

* MAP\_HANDLE representing the message’s property map.

## void IoTHubMessage\_Destroy(IOTHUB\_MESSAGE\_HANDLE ioTHubMessageHandle);

Disposes of resources allocated by the IoT Hub message.

### Arguments

|  |  |
| --- | --- |
| Name | Description |
| ioTHubMessageHandle | The handle created by a call to the create function. |

### Return

* No return.