Azure IoT Hub device management

library for C

The Azure IoT Hub device management (DM) library is part of the Azure IoT device SDK for C. This library helps connect resource-constrained devices to Azure IoT Hub device management. The library:

* Supports [CoAP](https://tools.ietf.org/html/rfc7252) over [TCP](https://datatracker.ietf.org/doc/draft-ietf-core-coap-tcp-tls/) (TLS support coming soon).
* Uses LWM2M objects, but provides high-level API abstractions so developers don't need a deep knowledge of LWM2M to do what they want.
* Supports authentication to Azure IoT Hub.
* Is written in ANSI C99 for portability to a variety of platforms.
* Includes low-level APIs that are useful on resource-constrained devices, or when operating more than one channel to IoT Hub in parallel (for example, a messaging channel and a device management channel).

# CoAP

IoT Hub device management uses CoAP over TCP as shown in the picture below:



CoAP over TCP/TLS is well suited for enterprise environments where UDP (the default transport for CoAP) traffic may be impeded (for example, by firewalls, or NAT hardware).

# What is Lightweight M2M?

Lightweight M2M is an application-level communications protocol from the Open Mobile Alliance designed for M2M and IoT device management. There is a video tutorial on LWM2M [here](https://www.youtube.com/watch?v=g-41ZdcTnXc) and slide deck presentation [here](http://www.slideshare.net/OpenMobileAlliance/oma-lwm2m-tutorial-by-arm-to-ietf-ace?qid=9a40eb2e-fb28-44f2-875c-8f3ad13f0cbc&v=default&b=&from_search=2).

# LWM2M Objects

The DM library uses LWM2M *objects* and *resources*. Objects describe the entities in the system (such as the device and the server), and *resources* describe the available properties and actions on those objects. The DM library exposes function callbacks for each resource on an object so that developers need only add device-specific logic to the callbacks without the need to understand LWM2M details. More on this later.



Some LWM2M objects found on a device are:

* **Device object:** Provides device-specific information that the service can read and, in some cases, update, such as manufacturer information, model number, serial number, firmware version, power sources, battery level, current time, time zone and UTC offset, free memory. It also defines several actions that the service can perform on a device, such as device reboot, and factory reset.
* **Firmware Update object**: Provides a firmware update action, which the service invokes. It also provides information such as the location of the firmware package and the status of any ongoing firmware update operation.
* **Server object:** Contains connection parameters and settings used for connecting devices to IoT Hub such as the lifetime of the registration and transport binding.
* **Location Object:** Reports the latitude, longitude, altitude, and velocity of a device.

Developers should have a good understanding of the device and firmware objects because the resources they contain map directly to the code developers must write for devices. To learn more about available LWM2M objects refer to [OMA LWM2M Object and resource registry](http://technical.openmobilealliance.org/Technical/technical-information/omna/lightweight-m2m-lwm2m-object-registry).

To illustrate the above concepts, the following table contains a partial list of the resources defined in the **Device** object:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resource Name | Allowed operation on resource | Is the resource mandatory or optional | Type of the resource data | Description of the resource |
| Manufacturer | Read | Optional | String | Manufacturer name |
| Firmware Version | Read | Optional | String | Current firmware version of device |
| Reboot | Execute | Mandatory |  | Reboot the LWM2M Device to restore the device from unexpected firmware failure. |
| Factory Reset | Execute | Optional |  | Perform factory reset of the device to make the device have the same configuration as at initial deployment. |
| Current Time | Read  Write | Optional | Time | Current UNIX time of the LWM2M Client.  The LWM2M Client should be responsible to increase this time value as every second elapses.  The LWM2M Server is able to write this Resource to make the LWM2M Client synchronized with the LWM2M Server. |

The sample applications included with the DM library source code each include a file called *device\_object.c* with functions that map to each of the resources in the **Device** object, like this:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CALLBACK HANDLERS

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
IOTHUB\_CLIENT\_RESULT on\_read\_device\_manufacturer(STRING\_HANDLE \*value)

{

\*value = /\* platform-specific code to get the value \*/;

return IOTHUB\_CLIENT\_OK;

}

IOTHUB\_CLIENT\_RESULT on\_read\_device\_firmwareversion(STRING\_HANDLE \*value)

{

\*value = /\* platform-specific code to get the value \*/;

return IOTHUB\_CLIENT\_OK;

}

IOTHUB\_CLIENT\_RESULT on\_execute\_device\_reboot()

{

\*value = /\* platform-specific code to get the value \*/;

return IOTHUB\_CLIENT\_OK;

}

IOTHUB\_CLIENT\_RESULT on\_execute\_device\_factoryreset()

{

\*value = /\* platform-specific code to get the value \*/;

return IOTHUB\_CLIENT\_OK;

}

IOTHUB\_CLIENT\_RESULT on\_read\_device\_currenttime(int \*value)

{

\*value = /\* platform-specific code to get the value \*/;

return IOTHUB\_CLIENT\_OK;

}

IOTHUB\_CLIENT\_RESULT on\_write\_device\_currenttime(int value)

{

\*value = /\* platform-specific code to get the value \*/;

return IOTHUB\_CLIENT\_OK;

}

When the DM library receives a request from IoT Hub to perform an operation on a resource, for example, to reboot the device or to report the device's firmware version, the library invokes the appropriate device callback function to complete the operation. The developer must implement all the relevant device callback functions, generally by writing platform-specific code.

IoT Hub can *observe* the value of any resource, which means that it can instruct the device to send periodic updates. It does this by making a **Write Attribute** request. Any object resource can have any or all of the following write attributes:

* Minimum Period – minimum observation quiet period, to limit notification frequency
* Maximum Period – maximum observation quiet period, to guarantee notifications
* Less than – low limit measurement notification, like low alarm, in engineering units
* Greater than – high limit measurement notification, like a high alarm, in engineering units
* Step – Minimum delta change required to notify, in engineering units
* Cancel – cancel observation on a resource

For the Private Preview release, the DM library will ignore any **Write Attribute** requests other than **Cancel**. This means that the server cannot (yet) configure the frequency of updates. Instead, the DM library simply updates the IoT Hub using a **Notify** request once every 5 seconds.

The observe/notify communication pattern described above is handled entirely between the DM library and IoT Hub. In other words, the device developer can focus on device-specific capabilities and leave data filtering and transmission logic to the DM library.

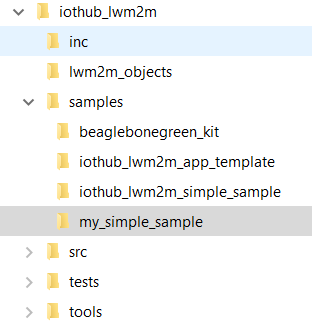
Note that the DM library does not cache object resource values. When the server reads a property value, the DM library calls into the client app (using the corresponding callback) to fulfill the request. Likewise, when the server sets a property value, the device management library calls into the client app to set the value.

The DM library does not expose APIs for sending telemetry data to the server at will. For that, developers should use the [IoTHub messaging APIs](https://github.com/Azure/azure-iot-sdks/tree/75107fa7b0e614a83dfcd81aff4727541d81fa28/c). In the future, IoT Hub will provide other communication channels that will enable devices to send urgent/high priority notifications messages to IoT Hub.

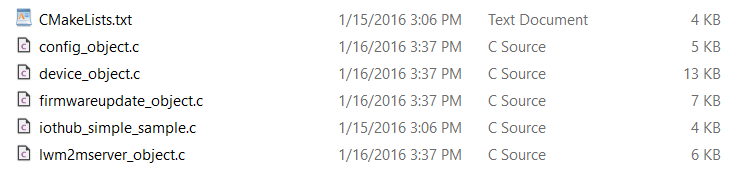
# Putting it all together

The Azure SDKs includes several IoT Hub device management samples.

Make a copy of the iotdm\_simple\_sample directory. Call the new directory "my\_ simple\_sample" as shown below:



Navigate to the **my\_simple\_sample** directory. The contents of the directory are:



The code for the **main** function in iotdm\_simple\_sample.c looks like this:

#include "iothub\_lwm2m.h"

// Other includes

static const char \*connectionString = "Add your connection string here"

int main(int argc, char \*argv[])

{

const char \*cs;

if (argc == 2) cs = argv[1];

else cs = connectionString;

// Open LWM2M channel

IOTHUB\_CHANNEL\_HANDLE hChannel;

if ((hChannel = IoTHubClient\_DM\_Open(cs, COAP\_TCPIP)) == NULL))

{

(void)printf("ERROR: IoTHubClient Channel Open returned NULL!\r\n")

}

else

{

// Add LWM2M objects: security, server, device firmware and config

register\_lwm2mserver\_object(hChannel);

register\_device\_object(hChannel);

register\_security\_object(hChannel);

register\_firwmareupdate\_object(hChannel);

register\_config\_object(hChannel);

// Start the agent and connect to the service

IOTHUB\_CLIENT\_RESULT rc;

if ((rc = IoTHubClient\_DM\_Start(hChannel)) != IOTHUB\_CLIENT\_OK))

{

(void)printf("ERROR: IoTHubClient\_DM\_Start() returned error %d\r\n", rc)

}

// Close channel

IoTHubClient\_DM\_Close(hChannel);

}

}

To register objects with the IoT Hub device management library, copy and paste the 'register\_<object name>' function found in the c file implementing each object. For example, for the **device object** use the register\_device\_object function found in the device\_object.c file. When the client application runs, the library code executes and invokes your object callbacks as appropriate. All that is left to do is to add your platform specific code into the object callbacks for the resources you want your device to support.

The next sections describe the process to implement the following common operations in your device:

* Report firmware information to the service
* Enable remote reboot capabilities
* Enable remote factory reset
* Enable firmware updates

Reporting device firmware information

Firmware information exists in the device object. You should edit the device\_object.c file where all the callbacks for the device object can be found. Locate the callback that maps to the device resource you want to report. In this example, the resource is **Firmware Version**. The table earlier in this document indicates that **Firmware Version** is a read-only resource. Locate the platform specific code for the function that maps to this resource. In this case the callback function is called on\_read\_device\_firmwareversion. You might also want to report information about the manufacturer, model number, and serial number using the same approach. Your code is invoked when the IoT Hub service wants to know about your firmware version, model number, manufacturer, and serial number. Your code might look similar to this:

STRING\_HANDLE propval\_device\_manufacturer = NULL;

STRING\_HANDLE propval\_device\_firmwareversion = NULL;

void set\_default\_device\_values()

{

propval\_device\_manufacturer = STRING\_construct("Contoso Corp.");

propval\_device\_firmwareversion = STRING\_construct("2.62");

}

IOTHUB\_CLIENT\_RESULT on\_read\_device\_manufacturer(STRING\_HANDLE \*value)

{

(void)printf("inside reader for Device\_Manufacturer\r\n");

printf("returning [%s]\r\n",STRING\_c\_str(propval\_device\_manufacturer));

STRING\_copy(\*value, STRING\_c\_str(propval\_device\_manufacturer));

return IOTHUB\_CLIENT\_OK;

}

IOTHUB\_CLIENT\_RESULT on\_read\_device\_firmwareversion(STRING\_HANDLE \*value)

{

(void)printf("inside reader for Device\_FirmwareVersion\r\n");

printf("returning [%s]\r\n",STRING\_c\_str(propval\_device\_firmwareversion));

STRING\_copy(\*value, STRING\_c\_str(propval\_device\_firmwareversion));

return IOTHUB\_CLIENT\_OK;

}

# Enabling remote reboot capabilities in your device

The reboot callback is also located in the device object, therefore you should look for the callback function that implements the reboot action in the device\_object.c file. This code is platform dependent. When IoT Hub instructs your device to reboot, this function in your code is called. Abstracting the platform dependent code, your callback function might look similar to this:

// Called when server calls reboot action

IOTHUB\_CLIENT\_RESULT on\_execute\_device\_reboot()

{

(void)printf("inside action handler for Device\_Reset\r\n");

return call\_os\_function\_to\_reboot\_device();

}

The factory reset functionality is very similar. The function you need to implement is listed below:

// Called when server calls the factory reset action

IOTHUB\_CLIENT\_RESULT on\_execute\_device\_factoryreset()

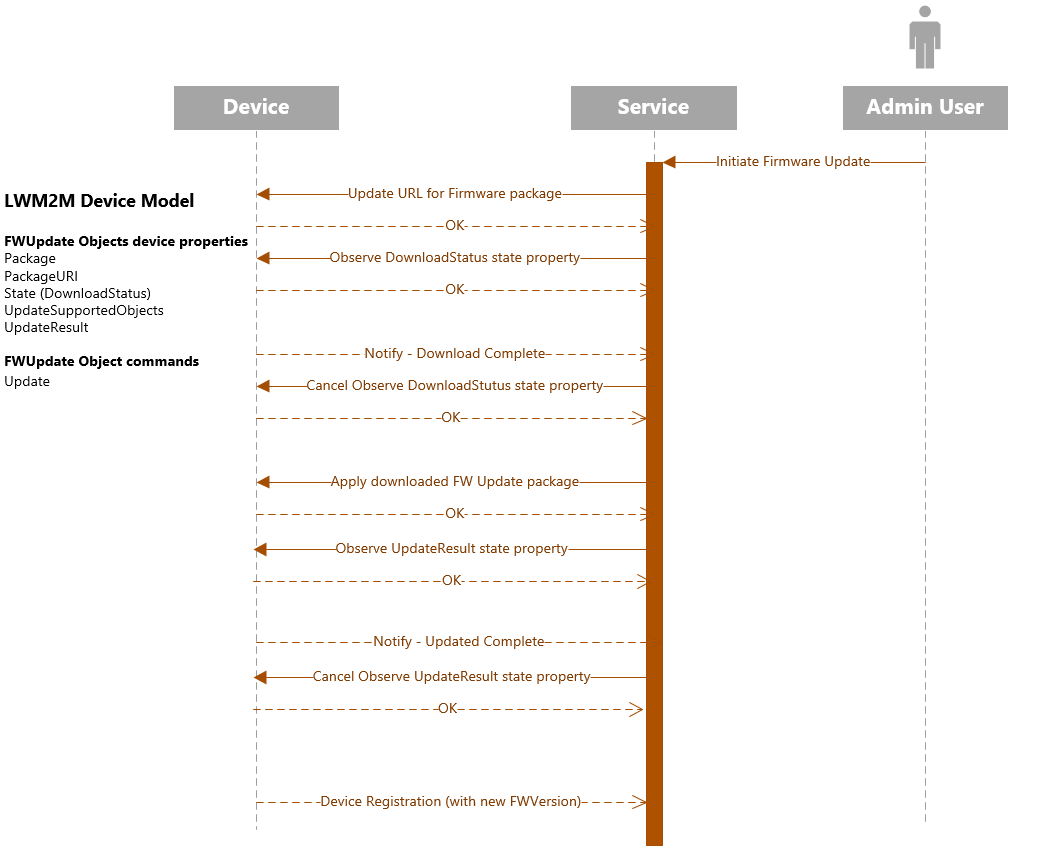
{

(void)printf("inside action handler for Device\_FactoryReset\r\n");

return call\_os\_function\_to\_do\_factoryreset();

}

# Enabling firmware updates in a device

The firmware update process is more complex than a factory reset or reboot. The flow of events on firmware updates is as follows: 

The device must support the flow of events shown in the diagram above. On the device, the code that implements the firmware update process is located in the file firmwareupdate\_object.c. Note that this code is platform specific.



To support firmware updates, you must implement the functions shown below:

// Called when server tells us to download a firmware package

IOTHUB\_CLIENT\_RESULT on\_write\_firmwareupdate\_packageuri(STRING\_HANDLE value)

{

use\_uri\_to\_start\_firmware\_download\_in\_background(STRING\_c\_str(value));

return IOTHUB\_CLIENT\_OK;

}

// Called when server wants to know the update state

IOTHUB\_CLIENT\_RESULT on\_read\_firmwareupdate\_state(int \*value)

{

if (background\_firmware\_download\_is\_in\_progress())

\*value = 2;

else if (background\_firmware\_download\_is\_complete())

\*value = 3;

else

\*value = 1;

return IOTHUB\_CLIENT\_OK;

}

// Called when server wants to know the result of the update

IOTHUB\_CLIENT\_RESULT on\_read\_firmwareupdate\_updateresult(int \*value)

{

if (update\_is\_complete())

\*value = 1;

else if (update\_failed())

\*value = get\_uppate\_error();

else

\*value = 0;

return IOTHUB\_CLIENT\_OK;

}

// Called when the server wants the client to perform the firmware update

IOTHUB\_CLIENT\_RESULT on\_execute\_firmwareupdate\_update ()

{

// Only valid to call if the server knows that the firmware is downloaded

ASSERT(background\_firmware\_update\_is\_complete());

perform\_firmware\_update();

return IOTHUB\_CLIENT\_OK;

}

One of the best ways to learn how to use the IoT Hub device management library is by looking at sample code. Below are three samples included in the SDK:

* simple\_sample
* 2channel\_sample
* Intel Edison sample

# Appendix A

Private Preview supported objects and resources

**Device object**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Resource Name | Resource ID | Operation | Type | Range and Units | Description |
| Manufacturer | 0 | Read | String |  | Manufacturer name |
| Model Number | 1 | Read | String |  | A model identifier (manufacturer specified string) |
| Device Type | 17 | Read | String |  | Type of device (manufacturer specified string)  Maps to service API: SystemPropertyNames.DeviceDescription |
| Serial Number | 2 | Read | String |  | Serial number of device |
| Firmware Version | 3 | Read | String |  | Current firmware version of the device  Maps to service API: SystemPropertyNames.HardwareVersion |
| Current Time | 13 | Read Write | Time |  | Current UNIX time of the device. The LWM2M Client should be responsible to increase this time value as every second elapses.  The LWM2M Server is able to write this Resource to make the LWM2M Client synchronized with the LWM2M Server. |
| Memory Free | 10 | Read | Integer | KB | Estimated current available memory of storage space which can store data and software in device |
| Memory Total | 21 | Read | Integer | KB | Total amount of storage space which can store data and software in the device |
| Battery Level | 9 | Read | Integer | 0-100% | Current battery level. Range: 0-100 |
| Battery Status | 20 | Read | Integer | 0-6 | Battery Status. Range 0-6 (Normal, Charging, Charge Complete, Damage, Low Battery, Not Installed, Unknown) |

**LWM2M Server object**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Resource Name | Resource ID | Operation | Type | Range and Units | Description |
| Default Minimum Period | 2 | Read Write | Integer | Seconds | Default value device should use for the minimum Period of Observation in the absence of this parameter being included in an observation  Maps to service API: SystemPropertyNames.DefaultMinPeriod |
| Default Maximum Period | 3 | Read Write | Integer | Seconds | Default value device should use for the maximum Period of Observation in the absence of this parameter being included in an observation  Maps to service API: SystemPropertyNames.DefaultMaxPeriod |
| Lifetime | 1 | Read Write | Integer |  | Specify the lifetime of the registration in seconds  Maps to service API: SystemPropertyNames.RegistrationLifetime |

**Firmware Update object**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Resource Name | Resource ID | Operation | Type | Range and Units | Description |
| Package | 0 | Write | Opaque |  | Firmware package |
| Package URI | 1 | Write | String | 0-255 bytes | URI from where the device can download the firmware package by an alternative mechanism. |
| Update | 2 | Execute |  |  | Updates Firmware by using the firmware package stored in Package, or, by using the firmware downloaded from the package URI. |
| State | 3 | Read | Integer | 1-3 | State of firmware update (Idle, Downloading, Downloaded)  Maps to service API: SystemPropertyNames.FirmwareUpdateState |
| Update Result | 5 | Read | Integer | 0-6 | Contains the results of downloading or updating firmware |
| PkgName | 6 | Read | String | 0-255 bytes | Name of Firmware Package |
| PackageVersion | 7 | Read | String | 0-255 bytes | Version of Firmware Package |

**Mechanism to control observation**

|  |  |  |
| --- | --- | --- |
| Observable Parameter | Description | Supported in Private Preview |
| Start | Start an observation | Yes |
| Stop | Stop an observation | No |
| P min | Minimum observation quiet period, to limit notification frequency | No |
| P max | Maximum observation quiet period, to guarantee notifications | No |
| Less than | Low limit measurement notification, like low alarm | No |
| Greater than | High limit measurement notification, like a high alarm | No |
| Step | Minimum delta change required to notify | No |