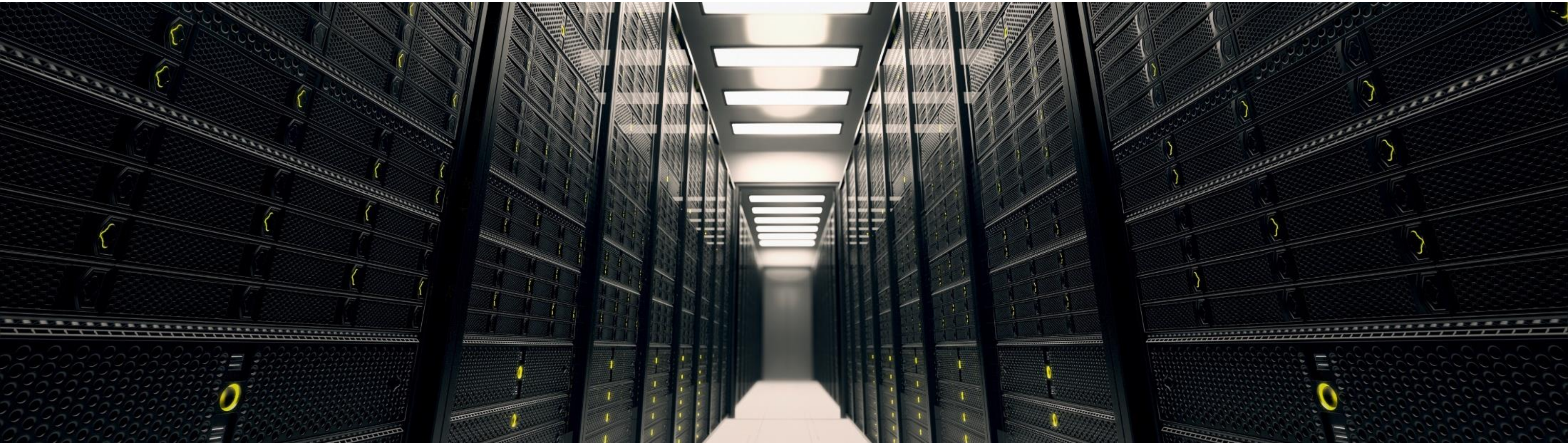
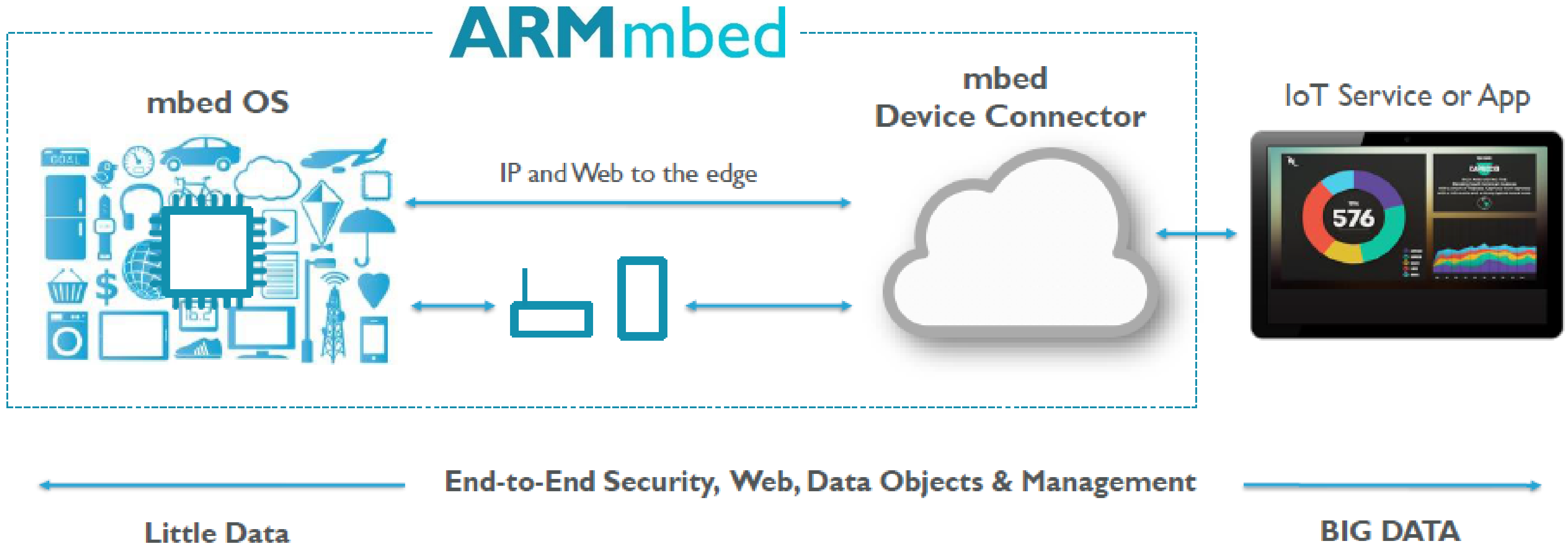
ARM mbed with us

Hands On – Getting to the Cloud



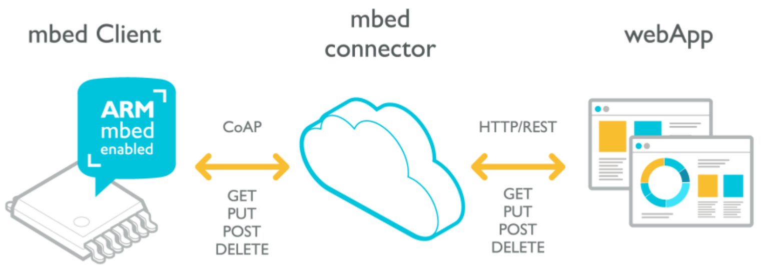
What is mbed Device Connector?

Server application that connects IoT devices with the web applications and services



mbed Device Connector

- mbed Device Connector is a service that lets you to provision and connect Internet of Things (IoT) devices to the cloud
- This service provides:
  - Secure end to end communication with the client SSL/TLS
  - Access to the resources shared from the client by web API



mbed Desktop Tools

ARM mbed Desktop Tools

- mbed CLI
  - Common interface across multiple compilers
  - Focused on ease of use, reproducibility
  - Open source project : https://github.com/ARMmbed/mbed-cli
  - Read the guide: https://github.com/ARMmbed/mbed-cli/blob/master/README.md

- Greentea
  - mbed OS test framework
  - Easy to execute tests on mbed Enabled devices

mbed CLI

- Command-line tool (Windows, Mac and Linux)
- Create or import applications
- Add/remove/update libraries
- Build apps & libraries
- Launch automated tests
- Generate IDE projects
- Publish code directly to mbed.org, github, others
  - No separate registry. Simplified dependency model.
- https://github.com/ARMmbed/mbed-cli

Invoked using "mbed" command

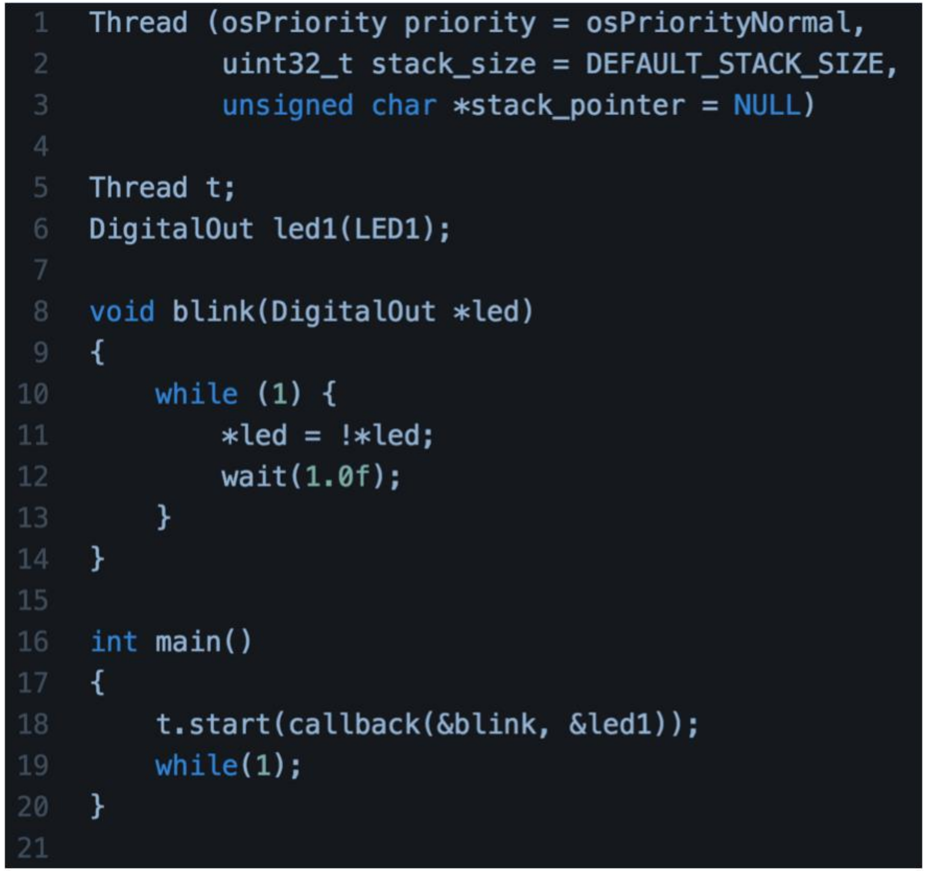
```
C:\>mbed compile
Building project mbed_blinky
Compile: main.cpp
Link: mbed_blinky
```

Compatible with mbed 2.0 (classic) and mbed OS 5.0 programs

mbed OS RTOS

mbed OS Threads

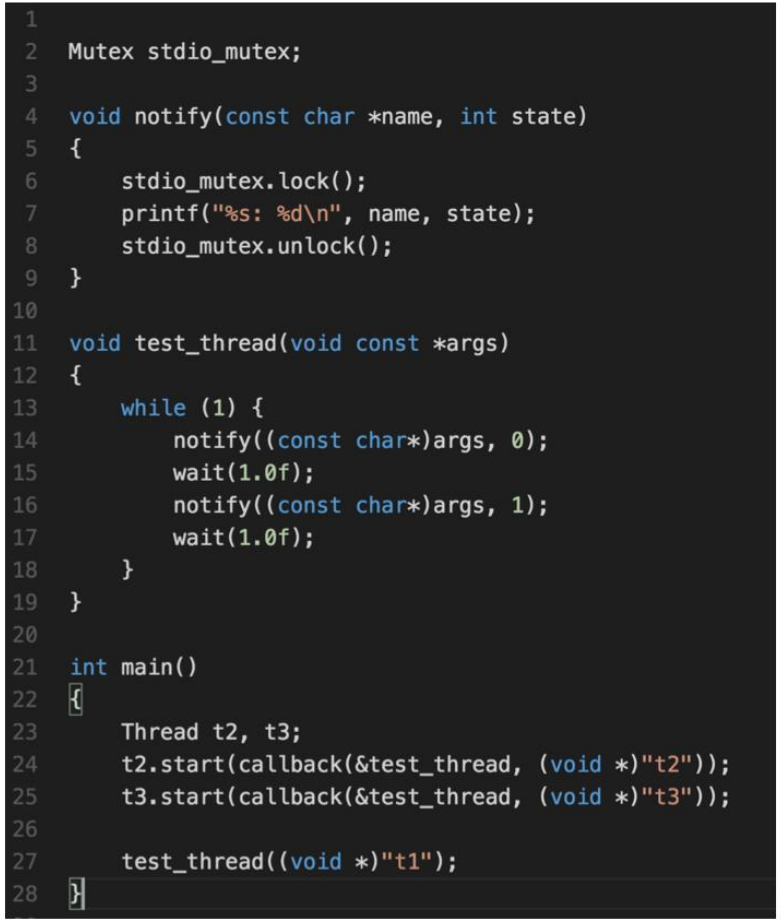
- Priorities
  - -3 ... 0 ... +3
  - osPriorityIdle ... osPriorityRealtime
- Stack
  - Dynamic allocation or user provided
- Signals
  - Thread::signal_set(int32_t)
  - Thread::signal_clear(int32_t)
- Delay
  - signal_wait(), wait(), yeild()



```
Thread (osPriority priority = osPriorityNormal,
        uint32_t stack_size = DEFAULT_STACK_SIZE,
        unsigned char *stack_pointer = NULL)

Thread t;
DigitalOut led1(LED1);

void blink(DigitalOut *led)
{
    while (1) {
        *led = !*led;
        wait(1.0f);
    }
}

int main()
{
    t.start(callback(&blink, &led1));
    while(1);
}
```

mbed OS Syncronization

- Synchronize access to shared resources
  - Mutex cannot be used from interrupt context!
- Mutex default timeout is osWaitForever
  - Mutex::lock(uint32_t ms)
  - Mutex::unlock()
- Semaphore default timeout is osWaitForever
  - Semaphore::wait(uint32_t ms)
  - Semaphore::release()

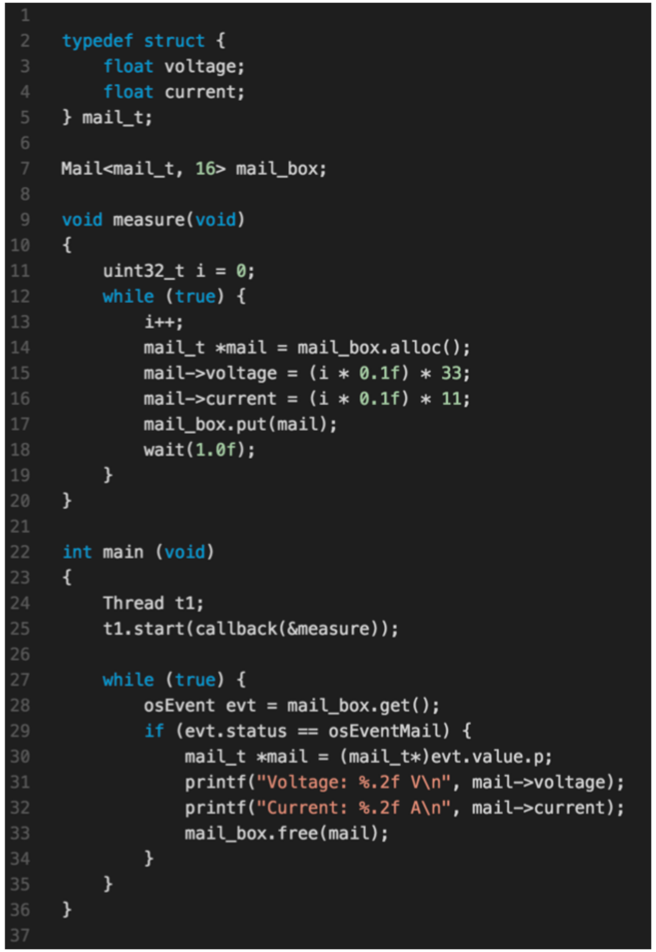


```

Mutex stdio_mutex;

void notify(const char *name, int state)
{
    stdio_mutex.lock();
    printf("%s: %d\n", name, state);
    stdio_mutex.unlock();
}

void test_thread(void const *args)
{
    while (1) {
        notify((const char*)args, 0);
        wait(1.0f);
        notify((const char*)args, 1);
        wait(1.0f);
    }
}

int main()
{
    Thread t2, t3;
    t2.start(callback(&test_thread, (void *)"t2"));
    t3.start(callback(&test_thread, (void *)"t3"));

    test_thread((void *)"t1");
}
```

AVNET SILICA

mbed OS Messages

- Queue is used for storing pointers to data
  - Queue<T, size>
  - Queue::put(T *)
  - Queue::get(osWaitForever)
- MemoryPool is used for data storage
  - MemoryPool<T, size>
  - MemoryPool::alloc() / calloc()
  - MemoryPool::free()
- Mail
  - Mail<T, size>
  - Managed Queue and MemoryPool

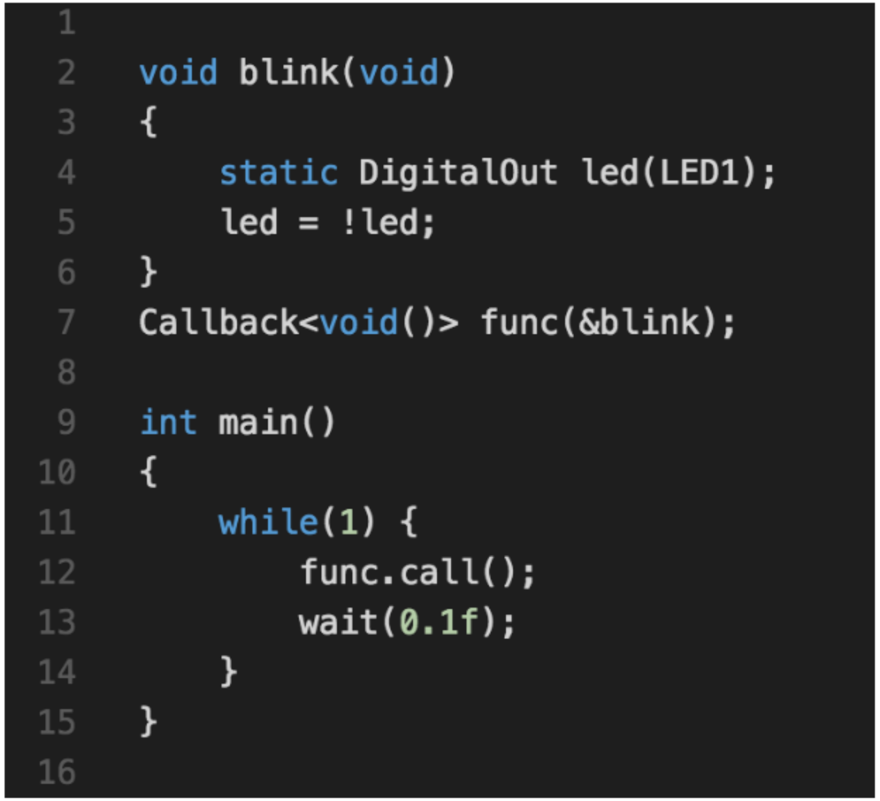


```

typedef struct {
    float voltage;
    float current;
} mail_t;

Mail<mail_t, 16> mail_box;

void measure(void)
{
    uint32_t i = 0;
    while (true) {
        i++;
        mail_t *mail = mail_box.alloc();
        mail->voltage = (i * 0.1f) * 33;
        mail->current = (i * 0.1f) * 11;
        mail_box.put(mail);
        wait(1.0f);
    }
}

int main (void)
{
    Thread t1;
    t1.start(callback(&measure));

    while (true) {
        osEvent evt = mail_box.get();
        if (evt.status == osEventMail) {
            mail_t *mail = (mail_t*)evt.value.p;
            printf("Voltage: %.2f V\n", mail->voltage);
            printf("Current: %.2f A\n", mail->current);
            mail_box.free(mail);
        }
    }
}

```

mbed OS events

mbed OS Callback

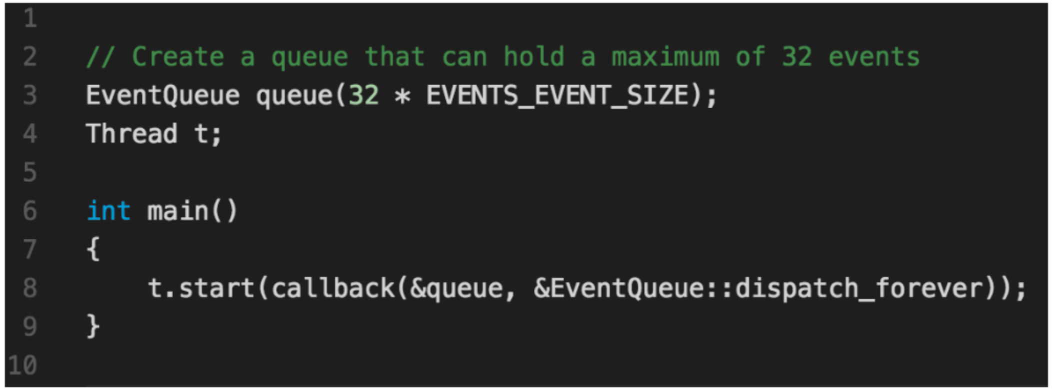
- Flexible function pointer
  - Object and member function
  - Any combination of return types and up to 5 parameters
  - C functions



```

void blink(void)
{
    static DigitalOut led(LED1);
    led = !led;
}
Callback<void()> func(&blink);

int main()
{
    while(1) {
        func.call();
        wait(0.1f);
    }
}

```

mbed OS EventQueue

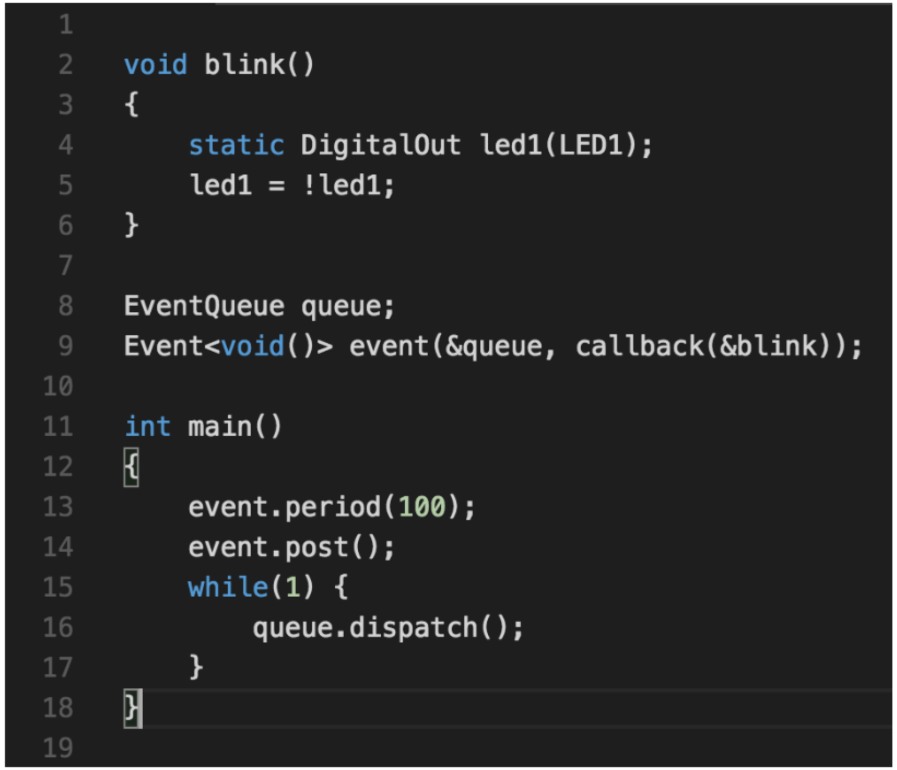
- Storage for many events
  - Events in an EventQueue are not pre-emptive
  - The queue is elastic until it runs out of memory though (plays catchup)
- RTOS aware – can have multiple queues at different priorities



```

// Create a queue that can hold a maximum of 32 events
EventQueue queue(32 * EVENTS_EVENT_SIZE);
Thread t;

int main()
{
    t.start(callback(&queue, &EventQueue::dispatch_forever));
}

```

mbed OS Event

- A Callback that is attached to an EventQueue
- Has attributes such as
  - Event::delay(int) – ms before dispatch
  - Event::period(int) – dispatch repetedly

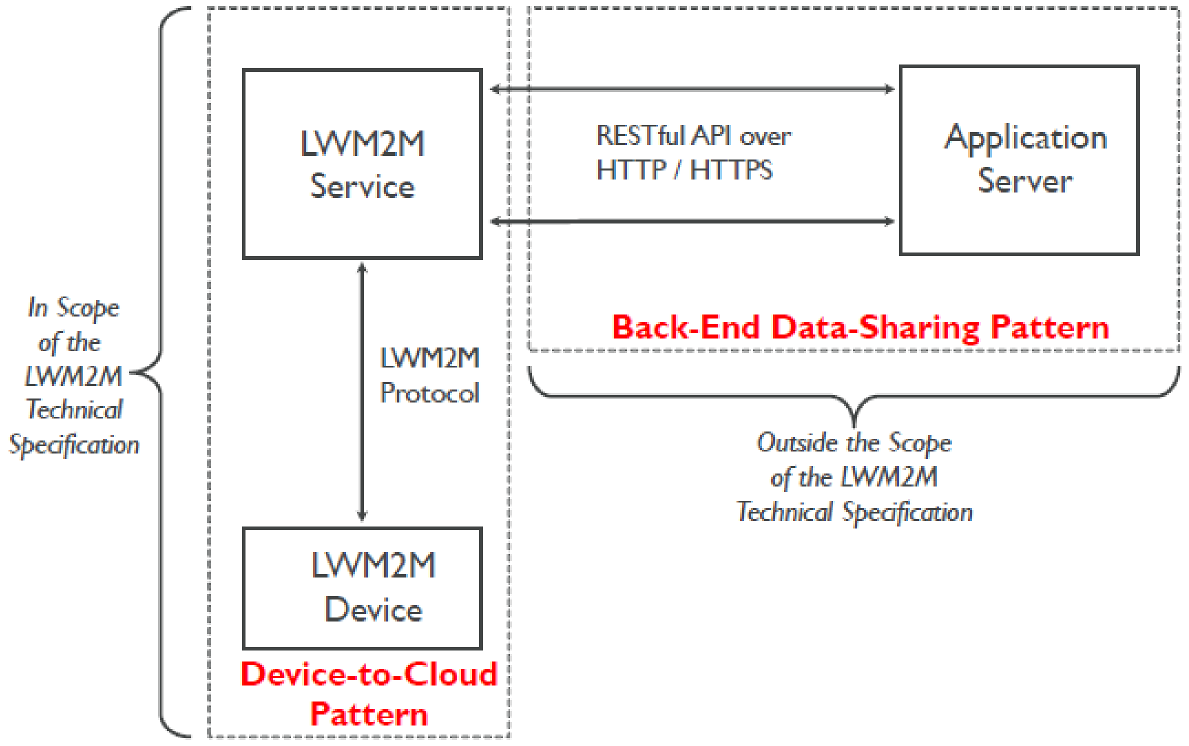


```

void blink()
{
    static DigitalOut led1(LED1);
    led1 = !led1;
}

EventQueue queue;
Event<void()> event(&queue, callback(&blink));

int main()
{
    event.period(100);
    event.post();
    while(1) {
        queue.dispatch();
    }
}

```

AVNET SILICA

Protocols and Standards

Architecture – 2 distinct domains


LWM2M Service
RESTful API over HTTP / HTTPS
Application Server
Back-End Data-Sharing Pattern
In Scope of the LWM2M Technical Specification
LWM2M Protocol
Outside the Scope of the LWM2M Technical Specification
LWM2M Device
Device-to-Cloud Pattern

LWM2M version 1.0 architecture

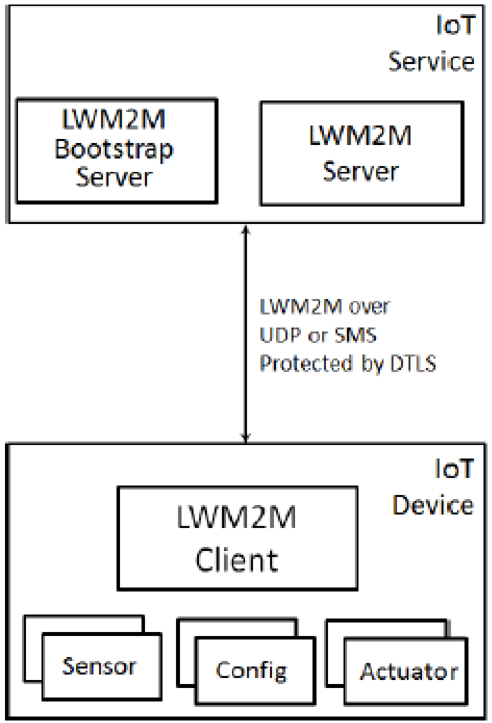


Figure 1: Entities in the LWM2M Architecture.

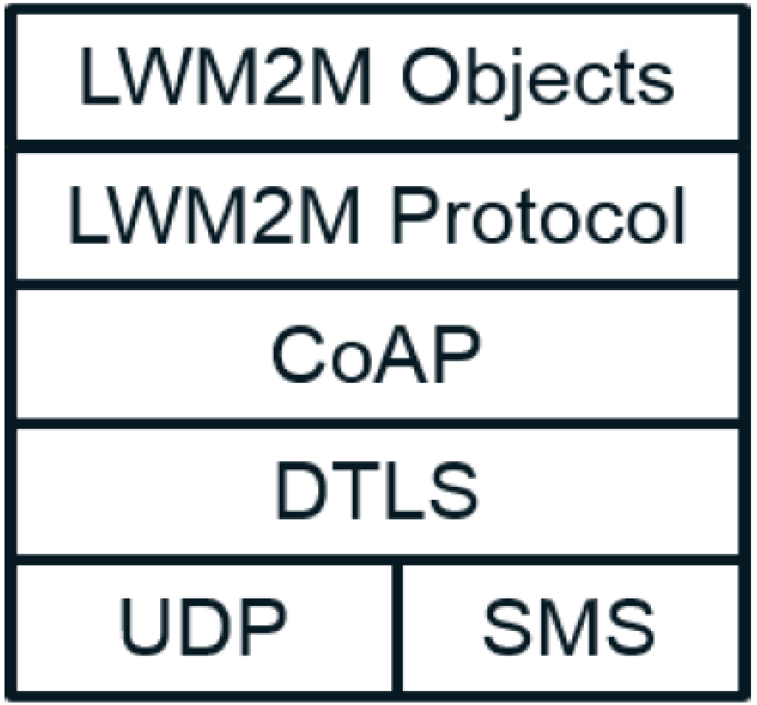


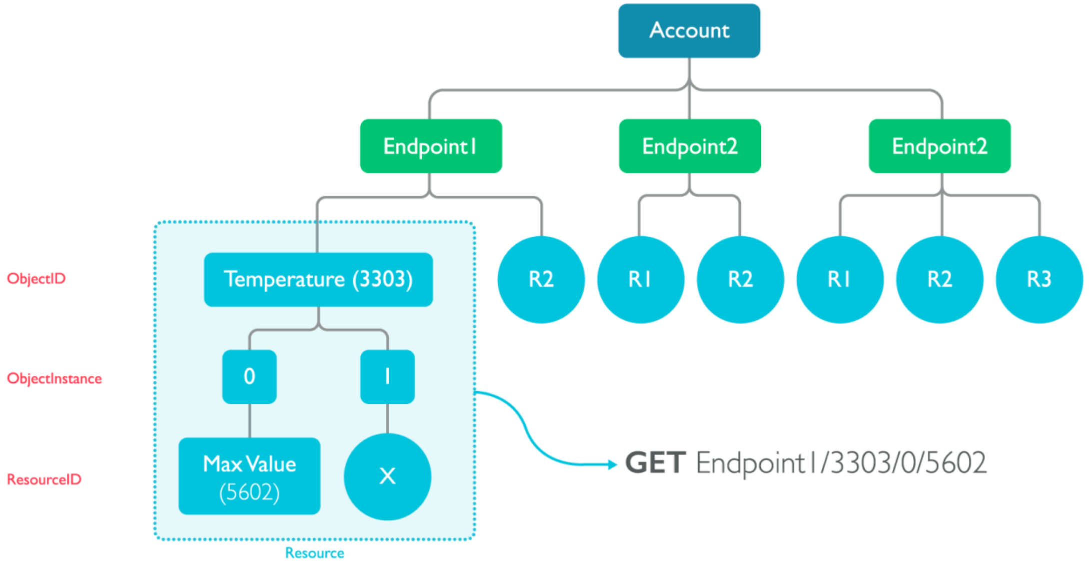
Figure 2: Protocol Stack

OMA Lightweight M2M

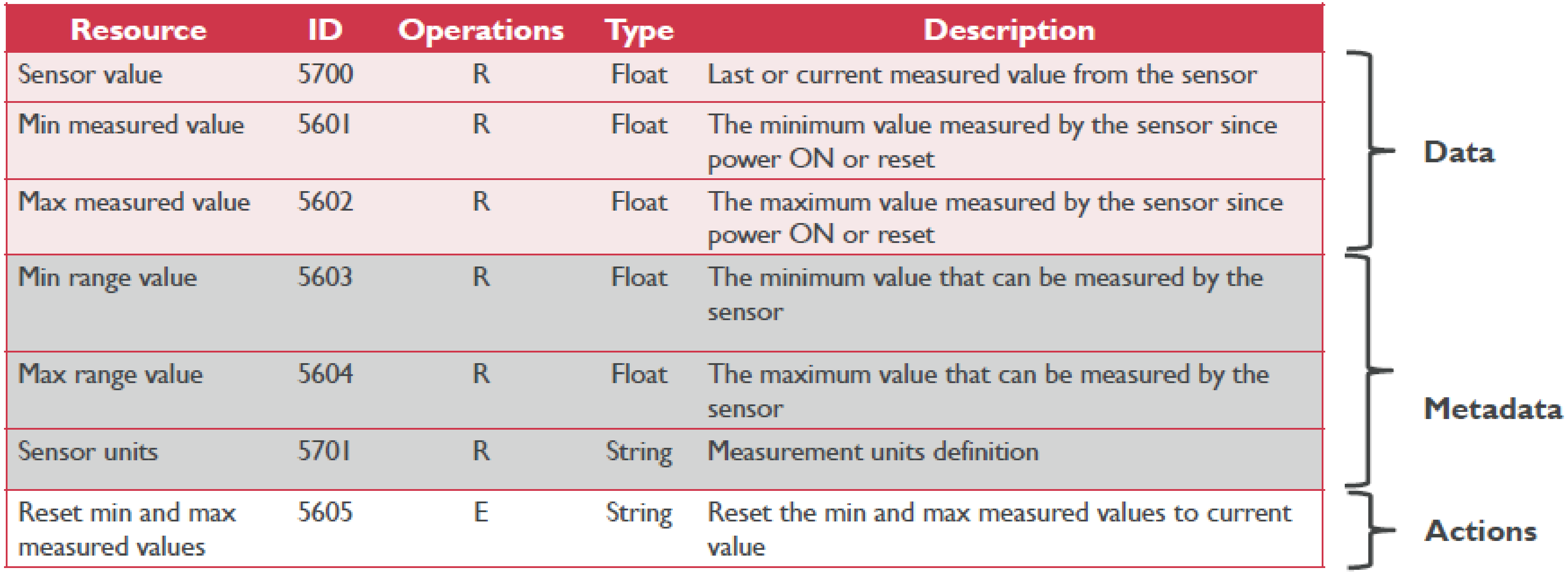
- LWM2M is a Device Management protocol optimized for IoT devices
- Manage IoT devices remotely, provision security credentials, and update over-the-air
- Standard protocols is the key in preventing vendor lock-in
  - Vendor lock-in -a customer dependent on a vendor for products and services, unable to use another vendor without substantial switching costs
- ARM is an active member in the OMA standard body activities
  - ARM client and server implementation are standard compliant
  - ARM participate in the on-going Test Fests computability activities

Generic information for LWM2M

- Hierarcial data structure
- {object ID}/{object instance ID}/{Resource ID}/{Resource instance ID}
- Resource instance level can be also omitted if not needed, which is the most common case
- {object ID}/{object instance ID} /{Resource ID}
- In ”human” –what is it / which instance of these? / what is the value of it
- 3303/0/5602
- In ”human”:temp sensor / 1st instance / max observed value



Example: IPSO Temperature object (/3333/0/5xxx)



Resource	ID	Operations	Type	Description
Sensor value	5700	R	Float	Last or current measured value from the sensor
Min measured value	5601	R	Float	The minimum value measured by the sensor since power ON or reset
Max measured value	5602	R	Float	The maximum value measured by the sensor since power ON or reset
Min range value	5603	R	Float	The minimum value that can be measured by the sensor
Max range value	5604	R	Float	The maximum value that can be measured by the sensor
Sensor units	5701	R	String	Measurement units definition
Reset min and max measured values	5605	E	String	Reset the min and max measured values to current value

Data

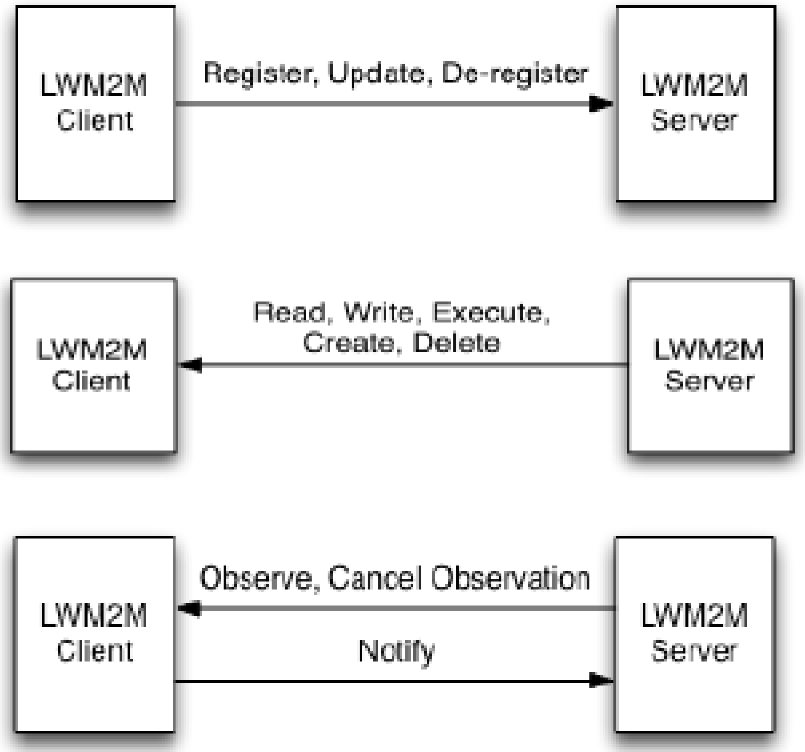
Metadata

Actions

Supported operations (LWM2M RESTful API)

High-level message pattern hiding details of networking protocols

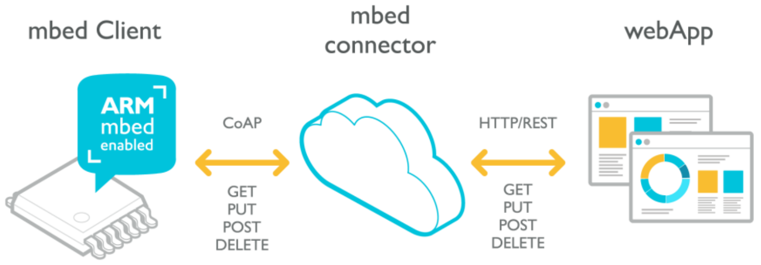
- Registration interface
  - Informs server about “existence” and supported functionalities (“I’m here, alive for 30 seconds, have temp sensor”)
- Device management & service enablement interface
  - Ability to access object instances and resources
- Information reporting interface
  - Subscribe/publish interaction for observing changes in resources


LWM2M Client
Register, Update, De-register
LWM2M Server
LWM2M Client
Read, Write, Execute, Create, Delete
LWM2M Server
LWM2M Client
Observe, Cancel Observation
Notify
LWM2M Server

Hands-On

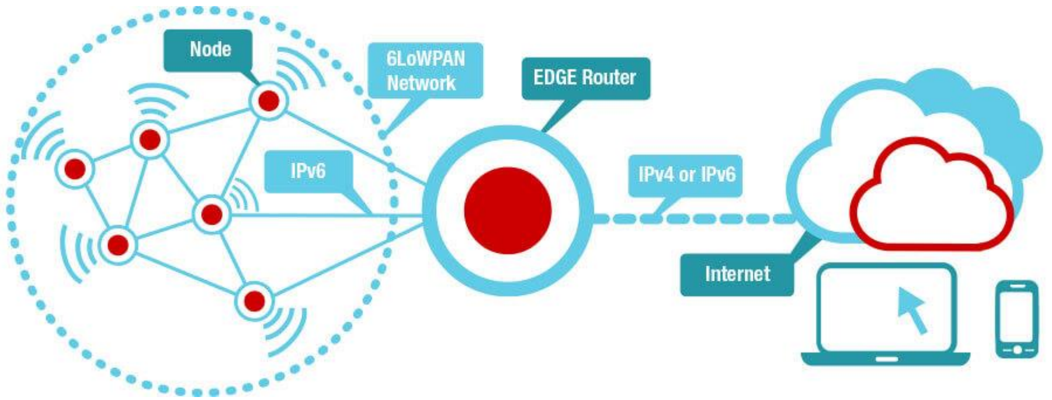
Our project

- Our project will connect the Sensor Board to the mbed Device Connector via a MESH network
- Initially we'll send a counter to mbed Device Connector and will be able to access to the value from a web browser
- In the next step we will modify the sources in order to send the temperature value coming from the ST HTS221 sensor



Network Topology

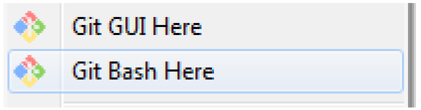
- The Sensor Node is a node of a MESH network
- Part of the network is an edge router which is connected to the cloud
- The connection from the Sensor Node to the EDGE Router (ST board NUCLEO_F429ZI) is 6LoWPAN sub-ghz wireless
- The EDGE Router is connected on internet via Ethernet cable
- The user can browse the mbed Device Connector website to access to the data of the nodes



Build the project (1/4)

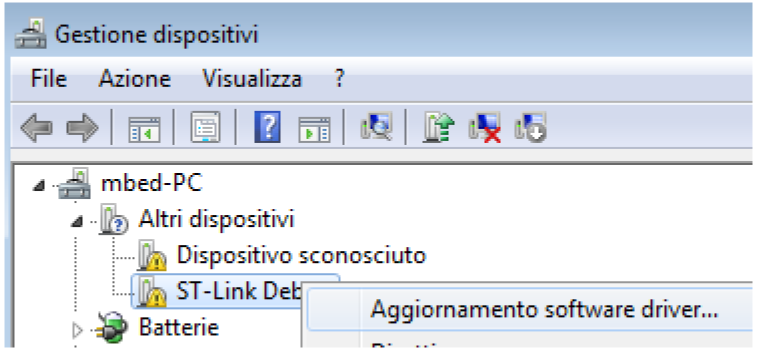
In the first part of the demo we will use the CLI (Command Line Interface)

- Open the Bash console in the folder where you want download the project
  - Press the right mouse button and select **Git Bash Here**



- Unzip the project from the *mbed-os-sensor-node.zip* file

- It will require few minutes, meanwhile start to setup your PC:
  - Connect the board to the PC
  - Install manually the drivers provided from the STSW-LINK009 zip file
    - Driver for ST-Link v2 (ST-Link Debug)
    - Driver for Virtual Com Port (unknown device)



Build the project (2/4)

- Setup the MAC address for the 6LoWPAN:
  - in the root directory of the project open the setting project file: mbed_app.json
  - you have to modify the “spirit1.mac-address” parameter. For the last three bytes substitute them with your birth date (this is to sure you have a unique MAC for the board in the demo network)

    For example:

    ```
    "spirit1.mac-address": "{0x7, 0x6, 0x5, 0x4, 0x3, MONTH, DAY, YEAR}“
    ```

    with “Feb/16/1980” will be:

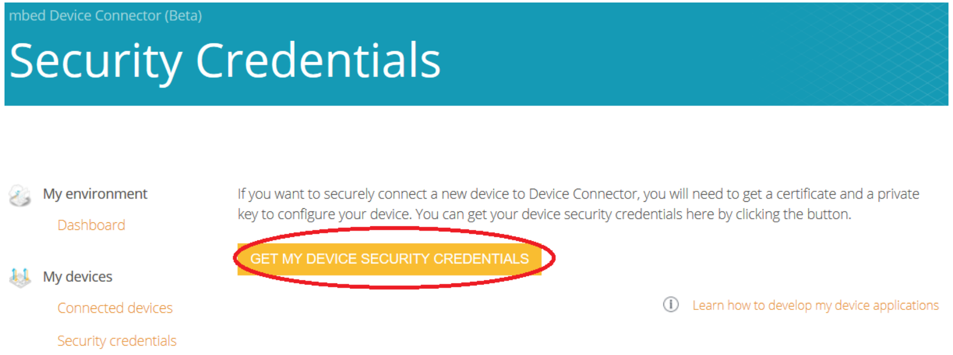
    ```
    "spirit1.mac-address": "{0x7, 0x6, 0x5, 0x4, 0x3, 0x02, 0x16, 0x80}"
    ```

Build the project (3/4)

Download from the ARM side the device certificate

- Login into your mbed Device Connector page

- Go to mbed Device Connector at the Security Credentials page:



mbed Device Connector (Beta)

Security Credentials

My environment
Dashboard
My devices
Connected devices
Security credentials

If you want to securely connect a new device to Device Connector, you will need to get a certificate and a private key to configure your device. You can get your device security credentials here by clicking the button.

GET MY DEVICE SECURITY CREDENTIALS
Learn how to develop my device applications

- In the root directory of the project open *security.h* file and edit replacing it with the text taken from the page which will open

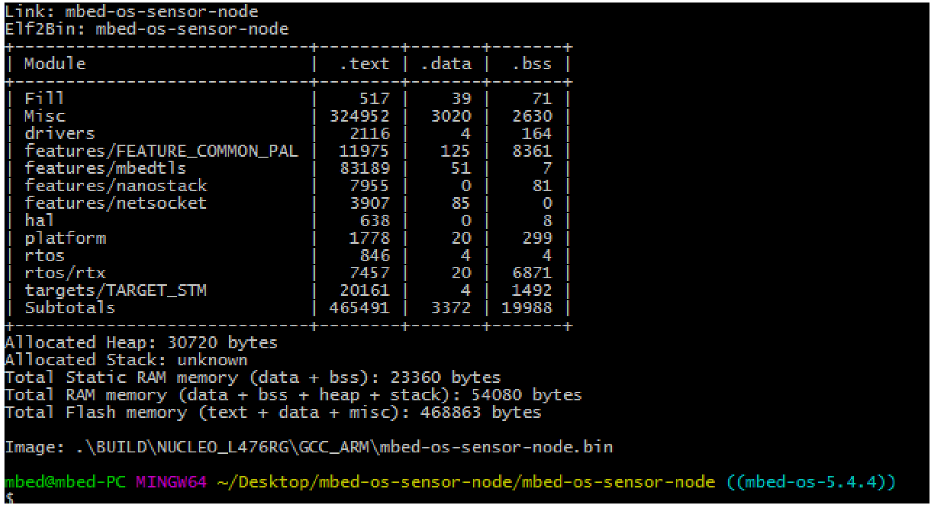
Build the project (4/4)

Project compilation using CLI

- Now all is ready to compile the project, enter in the mbed-os-sensor-node folder and type:

  mbed compile -m NUCLEO_L476RG -t GCC_ARM

- At the end of the compilation you will have a screen like this image



```
Link: mbed-os-sensor-node
Elf2Bin: mbed-os-sensor-node
+------------------------------+--------+-------+-------+
| Module                       |  .text | .data |  .bss |
+------------------------------+--------+-------+-------+
| Fill                         |    517 |    39 |    71 |
| Misc                         | 324952 |  3020 |  2630 |
| drivers                      |   2116 |     4 |   164 |
| features/FEATURE_COMMON_PAL  |  11975 |   125 |  8361 |
| features/mbedtls             |  83189 |    51 |     7 |
| features/nanostack           |   7955 |     0 |    81 |
| features/netsocket           |   3907 |    85 |     0 |
| hal                          |    638 |     0 |     8 |
| platform                     |   1778 |    20 |   299 |
| rtos                         |    846 |     4 |     4 |
| rtos/rtx                     |   7457 |    20 |  6871 |
| targets/TARGET_STM           |  20161 |     4 |  1492 |
| Subtotals                    | 465491 |  3372 | 19988 |
+------------------------------+--------+-------+-------+
Allocated Heap: 30720 bytes
Allocated Stack: unknown
Total Static RAM memory (data + bss): 23360 bytes
Total RAM memory (data + bss + heap + stack): 54080 bytes
Total Flash memory (text + data + misc): 468863 bytes

Image: .\BUILD\NUCLEO_L476RG\GCC_ARM\mbed-os-sensor-node.bin

mbed@mbed-PC MINGW64 ~/Desktop/mbed-os-sensor-node/mbed-os-sensor-node ((mbed-os-5.4.4))
$
```

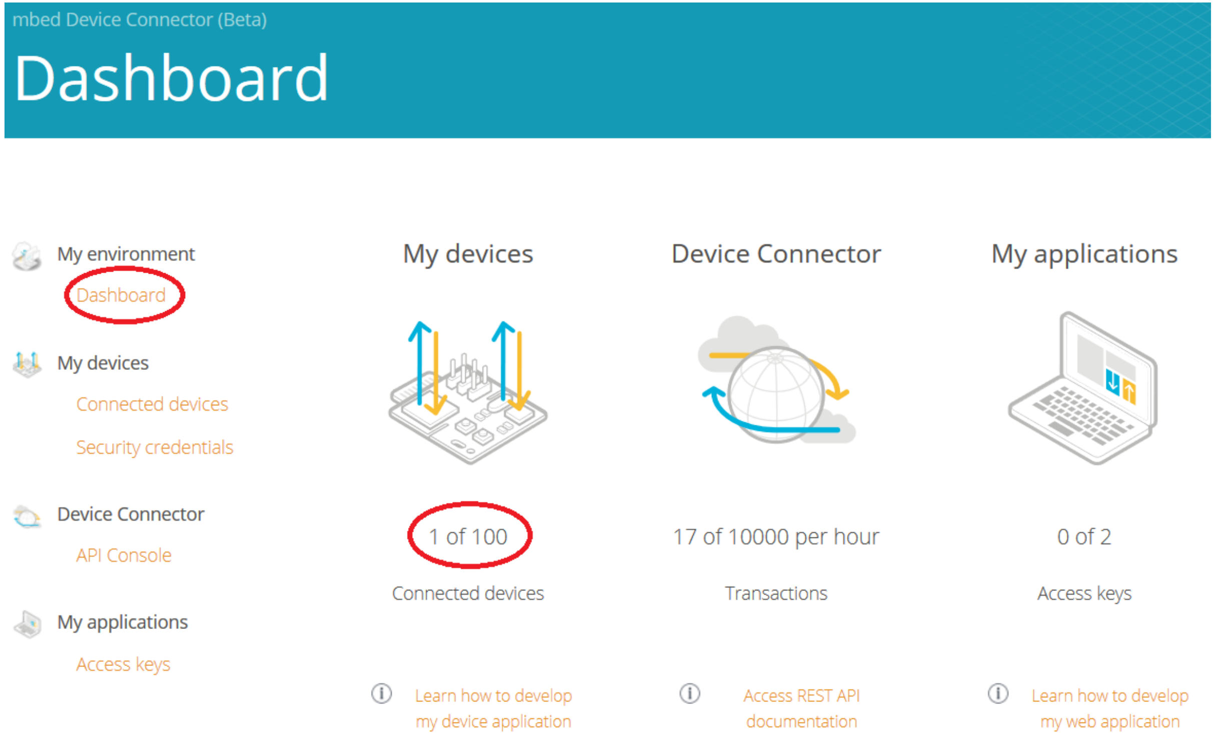
- The firmware will be created at:

  BUILD\NUCLEO_L476RG\GCC_ARM\mbed-os-sensor-node.bin

- Copy the binary file into the mass-storage of the board. Once it is copied the demo will start

Access to the resource (1/5)

- From the mbed Device Connector click on **Dashboard**
- Check whether your device is connected



mbed Device Connector (Beta)

Dashboard

My environment
- Dashboard

My devices
- Connected devices
- Security credentials

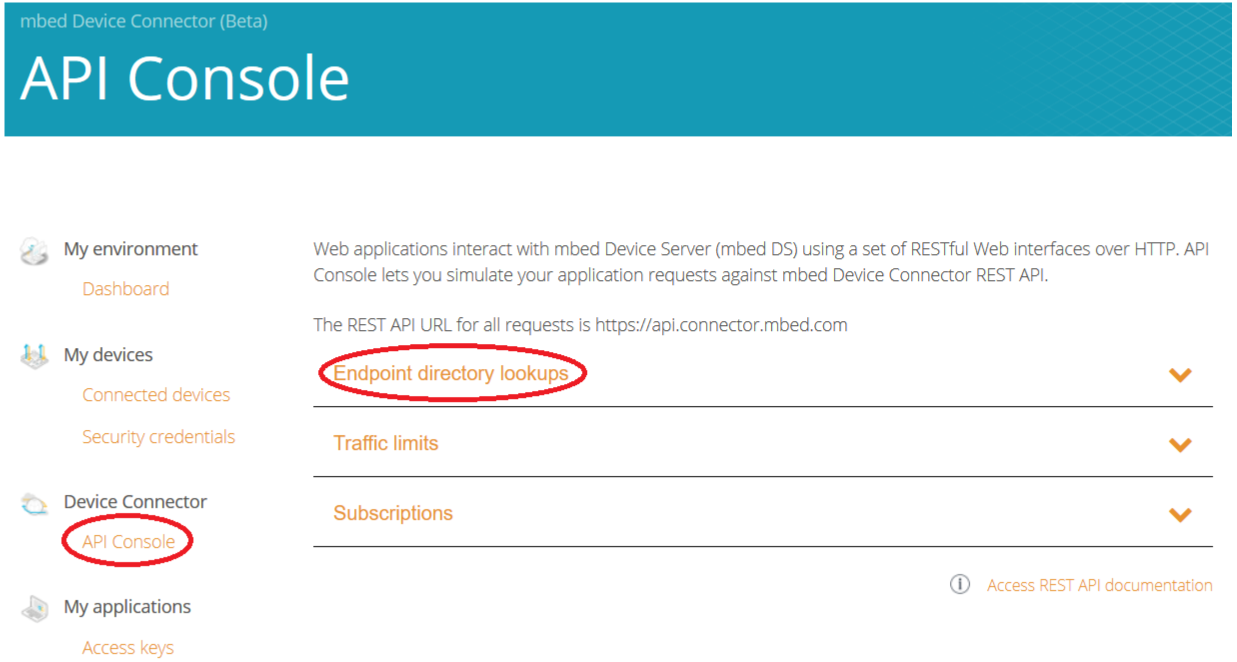
Device Connector
- API Console

My applications
- Access keys

My devices	Device Connector	My applications
1 of 100	17 of 10000 per hour	0 of 2
Connected devices	Transactions	Access keys
Learn how to develop my device application	Access REST API documentation	Learn how to develop my web application

Access to the resource (2/5)

– Click on **API Console** and then to **Endpoint directory lookups**


mbed Device Connector (Beta)

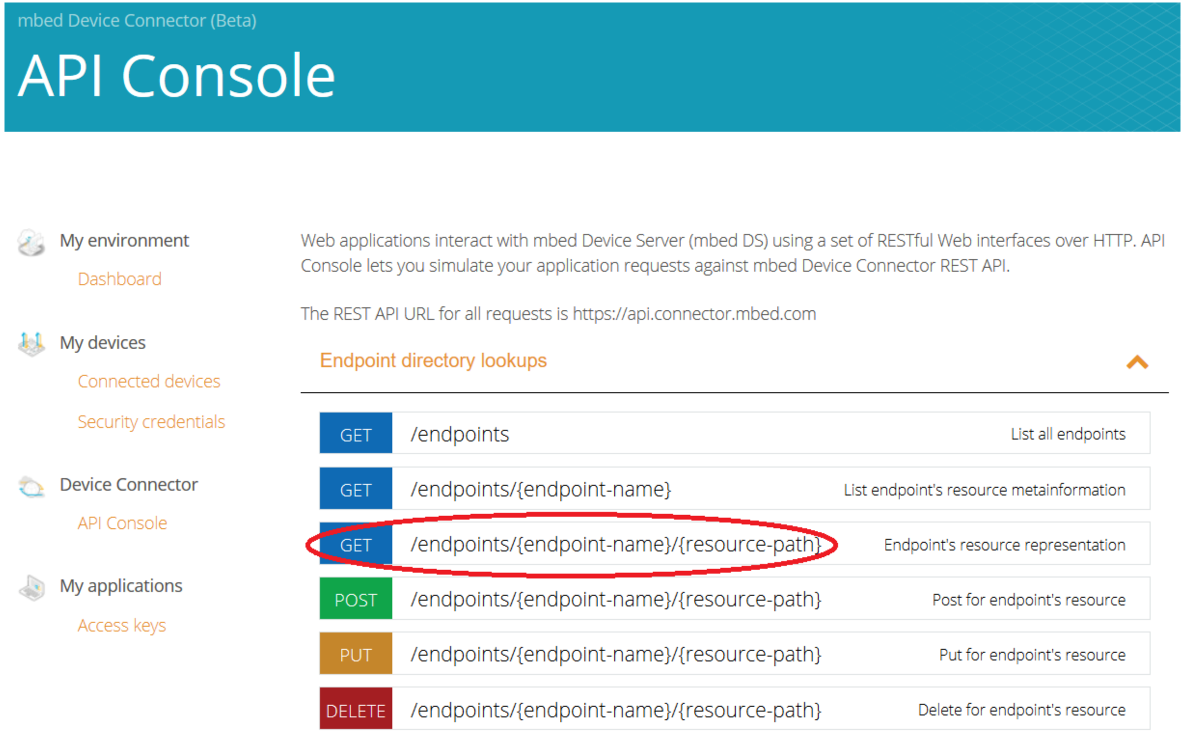
API Console

Web applications interact with mbed Device Server (mbed DS) using a set of RESTful Web interfaces over HTTP. API Console lets you simulate your application requests against mbed Device Connector REST API.

The REST API URL for all requests is https://api.connector.mbed.com

Access to the resource (3/5)

– Click on **GET Endpoint's resource representation**



mbed Device Connector (Beta)

API Console

- My environment
  - Dashboard
- My devices
  - Connected devices
  - Security credentials
- Device Connector
  - API Console
- My applications
  - Access keys

Web applications interact with mbed Device Server (mbed DS) using a set of RESTful Web interfaces over HTTP. API Console lets you simulate your application requests against mbed Device Connector REST API.

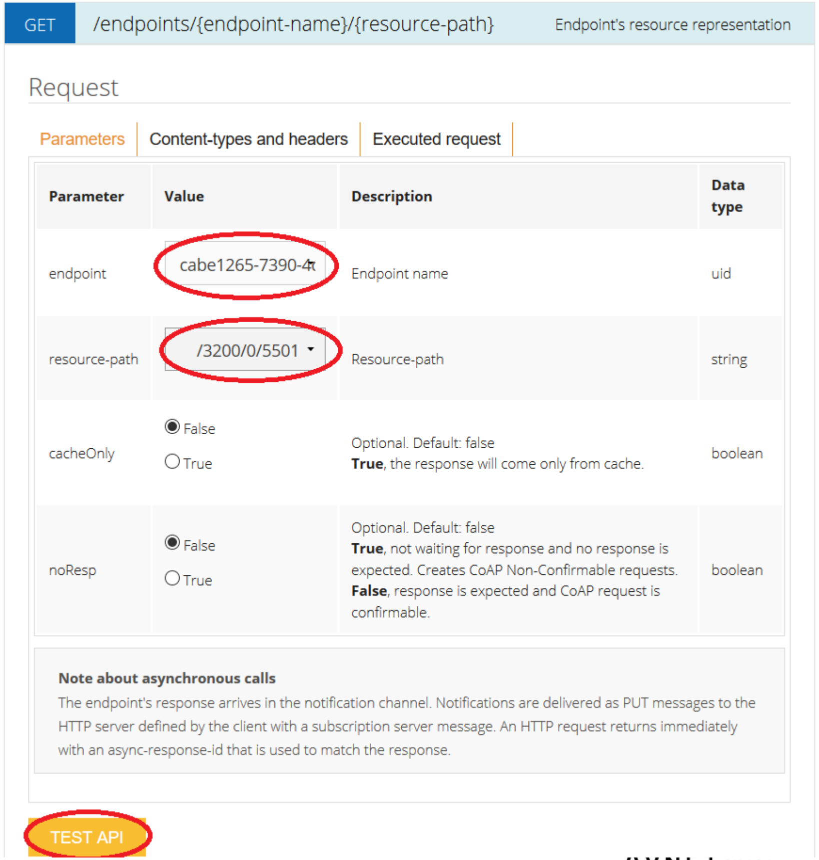
The REST API URL for all requests is https://api.connector.mbed.com

Endpoint directory lookups

GET	/endpoints	List all endpoints
GET	/endpoints/{endpoint-name}	List endpoint's resource metainformation
GET	/endpoints/{endpoint-name}/{resource-path}	Endpoint's resource representation
POST	/endpoints/{endpoint-name}/{resource-path}	Post for endpoint's resource
PUT	/endpoints/{endpoint-name}/{resource-path}	Put for endpoint's resource
DELETE	/endpoints/{endpoint-name}/{resource-path}	Delete for endpoint's resource

Access to the resource (4/5)

- From the GET form you have to select:
  - your endpoint
  - the resource you want read, in this case /3200/0/5501
- Then click on **TEST API** button



GET /endpoints/{endpoint-name}/{resource-path} Endpoint's resource representation

Request

Parameters | Content-types and headers | Executed request

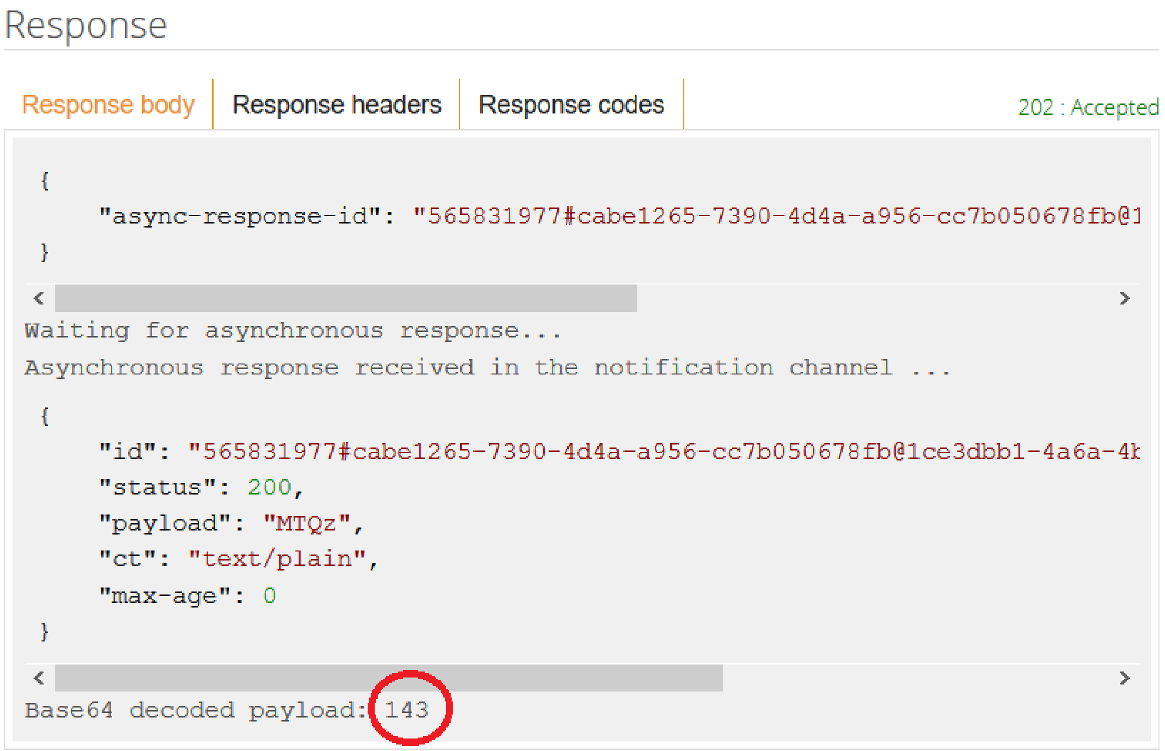
Parameter	Value	Description	Data type
endpoint	cabe1265-7390-4	Endpoint name	uid
resource-path	/3200/0/5501	Resource-path	string
cacheOnly	[✓ False] True	Optional. Default: false **True**, the response will come only from cache.	boolean
noResp	[✓ False] True	Optional. Default: false **True**, not waiting for response and no response is expected. Creates CoAP Non-Confirmable requests. **False**, response is expected and CoAP request is confirmable.	boolean

Note about asynchronous calls

The endpoint's response arrives in the notification channel. Notifications are delivered as PUT messages to the HTTP server defined by the client with a subscription server message. An HTTP request returns immediately with an async-response-id that is used to match the response.

Access to the resource (5/5)

– You will read the response decoded:



Response

Response body | Response headers | Response codes | 202 : Accepted

```
{
    "async-response-id": "565831977#cabe1265-7390-4d4a-a956-cc7b050678fb@1
}
```

```
Waiting for asynchronous response...
Asynchronous response received in the notification channel ...
```

```
{
    "id": "565831977#cabe1265-7390-4d4a-a956-cc7b050678fb@1ce3dbb1-4a6a-4k
    "status": 200,
    "payload": "MTQz",
    "ct": "text/plain",
    "max-age": 0
}
```

```
Base64 decoded payload: 143
```

AVNET SILICA

Hands on: Send the temperature

- The resourceID for read the sensor value has id **5700**. So if you want read the actual temperature from the endpoint1, first sensor, the command to read it will be:

  GET Endpoint1/**3303/0/5700**

- From the programming perspective we need modify the class **ButtonResource** in main.cpp:
  - From the constructor of the resource:
    - Create the ObjectID with the metod: M2MInterfaceFactory::create_object
    - Create the ResourceID with the metod: M2MInterfaceFactory::create_dynamic_resource, this resource is a float value
  - From the handler metod:
    - Get the right resource ID from the metod: M2MObjectInstance::resource
    - Insert in the buffer the temperature value taken from HTS211:: ReadTemp() function

- Compile, upload and check from the mbed Device Connector

Hands on: solution

- From the class **ButtonResource** in main.cpp:
  - From the constructor of the resource **ButtonResource(): counter(0) {** :
    - Create the ObjectID with the metod: M2MInterfaceFactory::create_object
      btn_object = M2MInterfaceFactory::create_object("3303");
    - Create the ResourceID with the metod: M2MInterfaceFactory::create_dynamic_resource
      M2MResource* btn_res = btn_inst->create_dynamic_resource("5700", "Temperature", M2MResourceInstance::FLOAT, true /* observable */);
  - From the handler metod **void handle_button_click()**:
    - Get the right resource ID from the metod: M2MObjectInstance::resource
      M2MResource* res = inst->resource("5700");
    - Insert in the buffer the temperature value taken from HTS211:: ReadTemp module
      int size = sprintf(buffer, "%f", hts221.ReadTemp());

Thank you!