Open Source Foundries

Zephyr™ Now Supports OMA LwM2M protocol! What Can It Do For Me?

Michael Scott
michael@opensourcefoundries.com

What can you expect during this presentation

- Background of the OMA Lightweight Machine-to-Machine (LwM2M). What is it and Why do I care?
- Discuss Zephyr's implementation of the LwM2M client:
  - LwM2M support in Zephyr™ by version: 1.9, 1.10 and 1.11
  - Brief demonstration of the LwM2M client sample using QEMU and Leshan Demo Server
  - Walk through some source code
  - Examine the current flash and RAM usage of the LwM2M engine and various other portions of the Zephyr™ stack
- Future plans for Zephyr™ LwM2M
- What is coming in the OMA LwM2M 1.1 Technical Specification update. (September 28, 2018 final release)

What is Lightweight Machine-to-Machine?

The Open Mobile Alliance Device Management working group created the LwM2M specification for the following reasons:

- Device management
- Simple resource model
- Based on CoAP (Constrained Application Protocol)
- Supports a REST API

Why not just use CoAP? Why have LwM2M?

- CoAP provides a request/response model between application endpoints which supports discovery of resources and services via a compact secure transmission protocol.
- **LwM2M provides a well defined set of interfaces and resources *not* defined by the CoAP specification.**
- Specifically, the LwM2M enabler specification ensures that most LwM2M client implementations should work with most LwM2M server implementations.

How does an LwM2M client work?

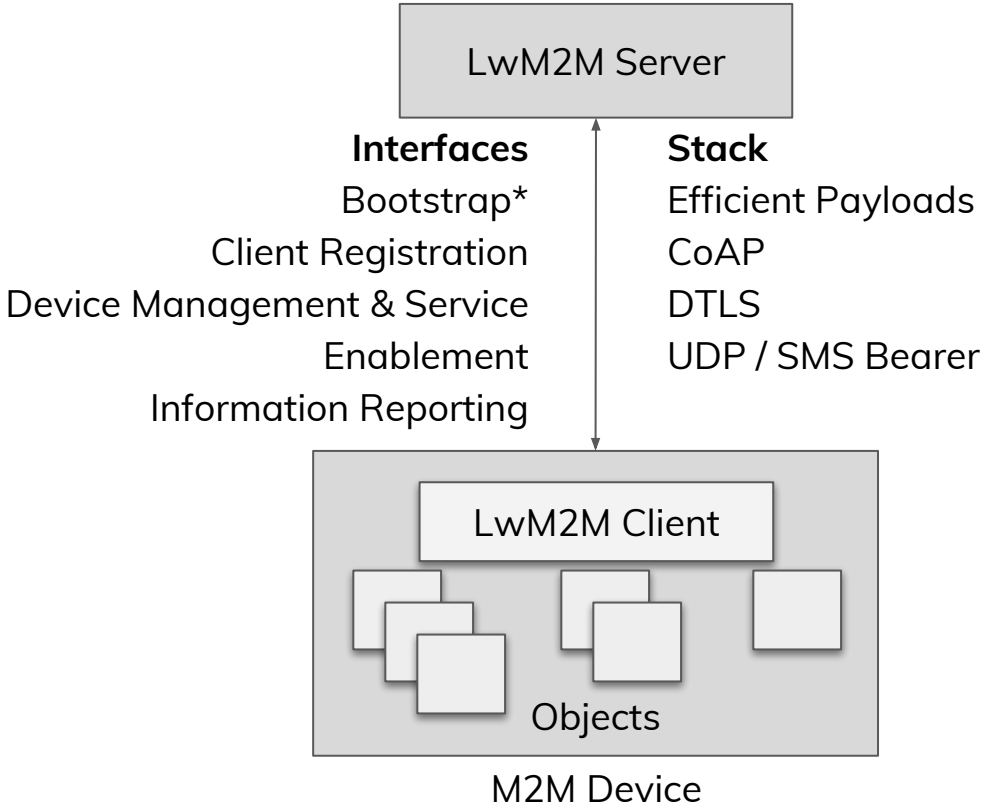
This question sounds obvious, but there are significant differences when we compare LwM2M clients to CoAP endpoints.

CoAP clients can manage resources made available on CoAP servers.

LwM2M clients run only on end devices. However, this “client” acts a lot more like a CoAP server when you look at how it works (seen in the next few slides).

LwM2M servers run in the cloud or on a local developer PCs and manage the well defined interfaces that connect the client to the server. There usually is a backend process which connects other services to the clients which have registered on the LwM2M server (think of a REST API). There is no equivalent in CoAP for the LwM2M server.

LwM2M Architecture Overview



"" items are not implemented yet in Zephyr™ as of today*

LwM2M Client / Server communication

"" items are not implemented yet in Zephyr™ as of today*

What are LwM2M objects?

- LwM2M objects are assigned an “object ID”. They can be single or multi-instance objects which would also require an “object instance ID” to access. Single instance objects normally only have a “0” instance.
- Some objects are marked as optional and don’t need to be implemented in every LwM2M client.
- LwM2M objects contain a list of predefined resources, each identified with a “resource ID”.
- A request for client resources use the URI Path options in the form of <object ID>/<object instance ID>/<resource ID>
  Example for device manufacturer resource: 3/0/0

What are the initial LwM2M enabler objects?

0 **LwM2M Security:** Keying material of a LwM2M Client enabling access to a specified LwM2M Server.
1 **LwM2M Server:** Information relating to the connection of an LwM2M server.
2 *Access Control: Used to check whether an LwM2M Server has access right for performing an operation.
3 **Device:** Device related information which can be queried by the LwM2M Server, and device reboot and factory reset functions.
4 *Connectivity Monitoring: Enables monitoring of parameters related to network connectivity.
5 Firmware Update: Management of firmware which is to be updated.
6 *Location: Where the LwM2M Client is located at the indicated point in time.
7 *Connectivity Statistics: Collection of statistical connectivity information.

is the object ID, **bold** are required for LwM2M clients and "*" items are not implemented yet in Zephyr™ as of today.

What does an LwM2M resource look like?

Each resource has a standard definition:

- Resource ID
- Name
- Operations (Read, Write, Execute)
- Single or Multi-instance
- Mandatory or optional for the LwM2M client to implement
- Data type (String, Date, Integer, etc)
- Possibly some information about acceptable range of values

LwM2M “Device” object definition as an example

Object definition

Name	Object ID	Instances	Mandatory	Object URN
Device	3	Single	Mandatory	urn:oma:lwm2m:oma:3

Resource definitions

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
0	Manufacturer	R	Single	Optional	String			Human readable manufacturer name
1	Model Number	R	Single	Optional	String			A model identifier (manufacturer specified string)
2	Serial Number	R	Single	Optional	String			Serial Number
3	Firmware Version	R	Single	Optional	String			Current firmware version of the Device. The Firmware Management function could rely on this resource.
4	Reboot	E	Single	Mandatory				Reboot the LwM2M Device to restore the Device from unexpected firmware failure.
5	Factory Reset	E	Single	Optional				Perform factory reset of the LwM2M Device to make the LwM2M Device to go through initial deployment sequence where provisioning and bootstrap sequence is

Other object and resource definitions for LwM2M

Beyond the initial objects defined by the LwM2M enabler specification (currently at v1.0.2: http://openmobilealliance.org/release/LightweightM2M/), there are many more object definitions which can be found in the “OMA LwM2M Object and Resource Registry”: http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html

They are identified by object IDs which fall into several ranges:

- 0 - 1023: Objects Produced by OMA
- 2048 - 10240: Objects Produced by 3rd party Standards Development Organizations
- 10241 -32768: Objects Produced by vendors or individuals
- 32769 - 42768: Bulk Objects Reserved by Vendors

IPSO Alliance: Smart Objects

Example: Light Control

Describes the state and functions of a typical piece of hardware acting as a light.

Light Control

Description

Description: This Object is used to control a light source, such as a LED or other light. It allows a light to be turned on or off and its dimmer setting to be control as a % between 0 and 100. An optional colour setting enables a string to be used to indicate the desired colour.

Object definition

Name	Object ID	Object Version	LWM2M Version
Light Control	3311		
Object URN		Instances	Mandatory
urn:oma:lwm2m:ext:3311		Multiple	Optional

Resource definitions

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
5850	On/Off	RW	Single	Mandatory	Boolean			This resource represents a light, which can be controlled, the setting of which is a Boolean value (1,0) where 1 is on and 0 is off.
5851	Dimmer	RW	Single	Optional	Integer	0-100	%	This resource represents a light dimmer setting, which has an Integer value between 0 and 100 as a percentage.
5852	On Time	RW	Single	Optional	Integer		sec	The time in seconds that the light has been on. Writing a value of 0 resets the counter.
5805	Cumulative active power	R	Single	Optional	Float		Wh	The total power in Wh that the light has used.
5820	Power Factor	R	Single	Optional	Float			The power factor of the light.
5706	Colour	RW	Single	Optional	String			A string representing a value in some color space
5701	Sensor Units	R	Single	Optional	String			If present, the type of sensor defined as the UCUM Unit Definition e.g. â€œCelâ€ for Temperature in Celcius.

Enter the Zephyr™ Project

“The Zephyr™ Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with security in mind.”

- Open Source RTOS! You can make a difference!
- Very fast paced development
- Supports building in Linux, MacOS and Windows
- IoT protocol support is getting better and better

Zephyr™ LwM2M client development for v1.9

- Initial submission of patches during July and August 2017
- LwM2M registration support using “/rd” resource of the LwM2M server to register client endpoint and perform periodic updates
- LwM2M enabler object support for Security, Server, Device and Firmware Update
- IPSO Temperature and Light Control smart objects
- Content format support for TLV, Plain Text and JSON (WRITE only)

Caveats:

- No Bootstrap or DTLS support
- Server and Security objects are not really used
- Packet sizes were hard coded to 384 bytes due to CoAP library limitations. This affected longer operations like DISCOVER where we could overrun that size.

Zephyr™ LwM2M client development for v1.10

- December 2017 release
- Many bug fixes from initial submission
- LwM2M engine optimizations
- Updated the firmware pull state machine to support coap2http proxy
- Callbacks added for registration client events (registration, update, disconnect, etc)
- Moved to a new multi-packet API for CoAP library. Packet sizes are back to 128 bytes and added on as needed
- And much much more

Caveats:

- No Bootstrap or DTLS support
- Server and Security objects are not really used yet

Zephyr™ LwM2M client development for v1.11

- March 2017 release (literally over the weekend)
- More bug fixes
- Added support for DTLS using PSK (Pre-shared Key) method
- Added WRITE_ATTRIBUTE support (can set observe timing on individual resources, etc)

Caveats:

- No Bootstrap support
- While DTLS support is in-place, the Security and Server objects are mainly unused. Bootstrap support will change this in the future.

Zephyr™ LwM2M engine design

“Using Zephyr’s LwM2M client allows you to have many full-featured objects without paying a huge penalty in binary size.”

This means:

- Objects use a registration API in the engine. They hand off data descriptions of the resources they use. This let’s the engine do most of the work!
- All non-required objects can be configured on or off at build time saving flash space
- Since the engine contains a lot of the boilerplate code, adding an object usually costs less than 1Kb of flash.
- User applications can then use engine callbacks to make alterations to these resources as they are read, written or executed.

Live Demo

Zephyr™ LwM2M client sample for QEMU

Instructions for LwM2M client setup:
http://docs.zephyrproject.org/samples/net/lwm2m_client/README.html

Requirements:

- Install Zephyr™ SDK v0.9.2
  - https://github.com/zephyrproject-rtos/meta-zephyr-sdk/releases/tag/0.9.2
- Leshan Demo Server (download and start)
  - wget https://hudson.eclipse.org/leshan/job/leshan/lastSuccessfulBuild/artifact/leshan-server-demo.jar
  - java -jar ./leshan-server-demo.jar -wp 8081
- Follow Getting Started Guide to clone Zephyr™ sources and install dependencies: http://docs.zephyrproject.org/getting_started/getting_started.html

Zephyr™ LwM2M client sample for QEMU (TL;DR)

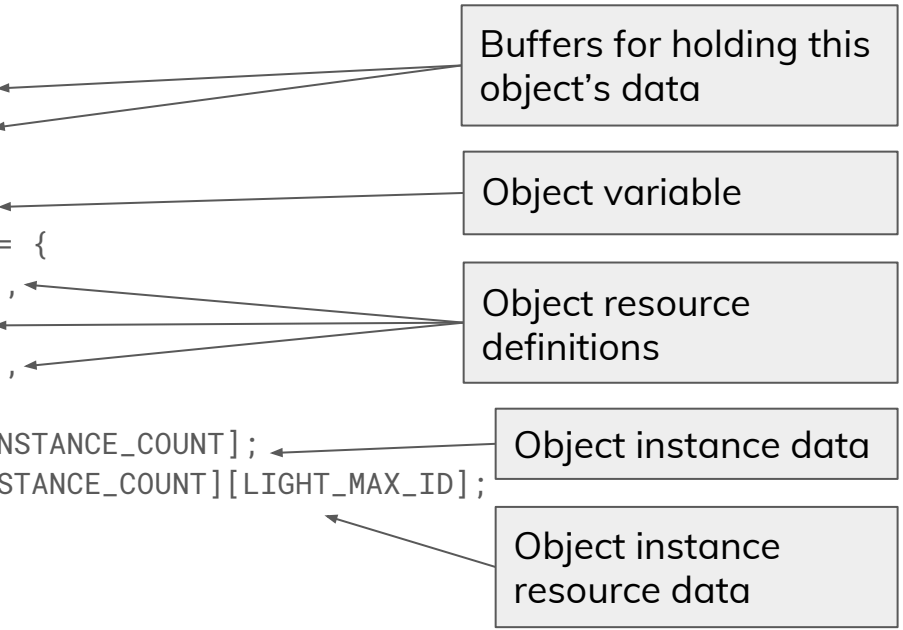
```
$ git clone git@github.com:zephyrproject-rtos/zephyr.git -b v1.11.0
$ cd zephyr
$ export ZEPHYR_SDK_INSTALL_DIR=<SDK Location>
$ export ZEPHYR_TOOLCHAIN_VARIANT=zephyr
$ . zephyr-env.sh
$ cd samples/net/lwm2m_client/
$ mkdir build && cd build
$ cmake -DBOARD=qemu_x86 ..
[...]
$ make run
```

Source Code

Let’s take a look at an LwM2M object

IPSO Light Control: subsys/net/lib/lwm2m/ipso_light_control.c

```
<includes>
<defines>
...
static bool on_off_value[MAX_INSTANCE_COUNT];
static u8_t dimmer_value[MAX_INSTANCE_COUNT];
...
static struct lwm2m_engine_obj light_control;
static struct lwm2m_engine_obj_field fields[] = {
     OBJ_FIELD_DATA(LIGHT_ON_OFF_ID, RW, BOOL),
     OBJ_FIELD_DATA(LIGHT_DIMMER_ID, RW, U8),
     OBJ_FIELD_DATA(LIGHT_ON_TIME_ID, RW, S32),
...
static struct lwm2m_engine_obj_inst inst[MAX_INSTANCE_COUNT];
static struct lwm2m_engine_res_inst res[MAX_INSTANCE_COUNT][LIGHT_MAX_ID];
```



Let's take a look at an LwM2M object

IPSO Light Control: subsys/net/lib/lwm2m/ipso_light_control.c

Object instance creation function

```
static struct lwm2m_engine_obj_inst *light_control_create(u16_t obj_inst_id)
{
<avoid duplicate instance checks and calculate the "next index" as "avail">
...
on_off_value[avail] = false;
dimmer_value[avail] = 0;
...
INIT_OBJ_RES_DATA(res[avail], i, LIGHT_ON_OFF_ID, &on_off_value[avail], sizeof(*on_off_value));
INIT_OBJ_RES_DATA(res[avail], i, LIGHT_DIMMER_ID, &dimmer_value[avail], sizeof(*dimmer_value));
...
inst[avail].resources = res[avail];
inst[avail].resource_count = i;
...
return &inst[avail];
}
```

Set default instance values

Assign buffers to resource elements

Assign resources to instance and set count

Return the new instance object

Let’s take a look at an LwM2M object

IPSO Light Control: subsys/net/lib/lwm2m/ipso_light_control.c

Object registration function

```
static int ipso_light_control_init(struct device *dev)
{
  int ret = 0;

  memset(inst, 0, sizeof(*inst) * MAX_INSTANCE_COUNT);
  memset(res, 0, sizeof(struct lwm2m_engine_res_inst) * MAX_INSTANCE_COUNT * LIGHT_MAX_ID);

  light_control.obj_id = IPSO_OBJECT_LIGHT_CONTROL_ID;
  light_control.fields = fields;
  light_control.field_count = sizeof(fields) / sizeof(*fields);
  light_control.max_instance_count = MAX_INSTANCE_COUNT;
  light_control.create_cb = light_control_create;
  lwm2m_register_obj(&light_control);

  return ret;
}

SYS_INIT(ipso_light_control_init, APPLICATION,CONFIG_KERNEL_INIT_PRIORITY_DEFAULT);
```

SYS_INIT functions start automatically

Let’s look at the LwM2M client sample application

samples/net/lwm2m_client/src/lwm2m-client.c

```
<includes>
<defines like CLIENT_MANUFACTURER and CLIENT_MODEL_NUMBER>
...
static struct device *led_dev;
static u32_t led_state;
static struct lwm2m_ctx client;
...
static int led_on_off_cb(u16_t obj_inst_id, u8_t *data, u16_t data_len,
                         bool last_block, size_t total_size)
{
<callback code to toggle LED depending on led_state>
}

static int device_reboot_cb(u16_t obj_inst_id)
{
<callback code to reboot the device (or simulated it in this case)>
}
```

Let’s look at the LwM2M client sample application

samples/net/lwm2m_client/src/lwm2m-client.c

```
static int lwm2m_setup(void)
{
...
  lwm2m_engine_set_string("3/0/0", CLIENT_MANUFACTURER);
  lwm2m_engine_set_string("3/0/1", CLIENT_MODEL_NUMBER);
...
  lwm2m_engine_register_exec_callback("3/0/4", device_reboot_cb);
  lwm2m_engine_register_exec_callback("3/0/5", device_factory_default_cb);
...
  if (init_led_device() == 0) {
    lwm2m_engine_create_obj_inst("3311/0");
    lwm2m_engine_register_post_write_callback("3311/0/5850", led_on_off_cb);
  }
  return 0;
}
```

Let’s look at the LwM2M client sample application

samples/net/lwm2m_client/src/lwm2m-client.c

```
static void rd_client_event(struct lwm2m_ctx *client, enum lwm2m_rd_client_event client_event)
{
  switch (client_event) {
  case LWM2M_RD_CLIENT_EVENT_NONE:
    break;
  ...
  case LWM2M_RD_CLIENT_EVENT_REGISTRATION_FAILURE:
    SYS_LOG_DBG("Registration failure!");
    break;
  case LWM2M_RD_CLIENT_EVENT_REGISTRATION_COMPLETE:
    SYS_LOG_DBG("Registration complete");
    break;
  case LWM2M_RD_CLIENT_EVENT_REG_UPDATE_FAILURE:
    SYS_LOG_DBG("Registration update failure!");
    break;
  ...
  }
}
```

Let’s look at the LwM2M client sample application

samples/net/lwm2m_client/src/lwm2m-client.c

```
void main(void)
{
  ...
  memset(&client, 0x0, sizeof(client));
  client.net_init_timeout = WAIT_TIME;
  client.net_timeout = CONNECT_TIME;
  ...
  ret = lwm2m_rd_client_start(&client, CONFIG_NET_APP_PEER_IPV6_ADDR,
                              CONFIG_LWM2M_PEER_PORT, CONFIG_BOARD,
                              rd_client_event);
  ...
}
```

Another Live Demo: Actual Hardware

Example of a Smart Light using Zephyr™ LwM2M client

- Examine the device connected to our local Leshan Demo Server
- Found a bug! Fix it and perform an over the air update

Example of an LTE connected IoT device

- Examine the device connected to a public Leshan Demo Server over LTE.

What are the hardware requirements?

Estimated flash and RAM sizes for various portions of the Zephyr™ kernel:

- 29Kb flash / 11Kb RAM: LwM2M subsys (client, objects and encoders) + CoAP + HTTP (url parsing) + net_app libraries
- 29Kb flash / 12Kb RAM: IP stack configured for UDP only
- 64Kb flash / 12Kb RAM: Bluetooth stack (host and controller)
- **35Kb flash / 21Kb RAM: MbedTLS/tinycrypt when DTLS security enabled! Not cheap!**
- 35Kb flash / ~8Kb RAM: Basic kernel support when using LwM2M client such as minimal libc, memory management, FIFO support and userspace / stack protection.
- Might have other drivers such as LED, ethernet, wifi or 802.15.4 support.

Example: Nordic™ nRF52832 w/ 512K flash and 64K RAM with a 32K flash space for mcuboot and 2 Zephyr™ kernel banks of 212K, the LwM2M client fits w/ DTLS security.

What's next for Zephyr™ beyond v1.11 release?

- More engine optimizations to reduce the flash and RAM footprints
- Bootstrap support
- Add more LwM2M enabler objects:
  - Access Control
  - Connectivity Monitoring
  - Connectivity Stats
- More object support! Lots to choose from:
  - Accelerometer
  - On/Off Switch
  - Push Button
  - Many many more!

What's next in the OMA LwM2M v1.1 update?

Final Release: September 28, 2018

- CoAP over TCP/TLS to enable LwM2M to be used in networks with restrictive firewalls is another possible use case.
- CBOR encoding format (RFC 7049) is being evaluated.
- DTLS / TLS profiles for IoT (RFC 7925) is on the list of desired functionality.
- Extensions to fully utilize new low power WAN technologies such as **LTE-M**, NB-IoT and LoRa. (Objects for management of NB-IoT connectivity have already been defined.)

Source:
http://openmobilealliance.org/iot/lightweight-m2m-lwm2m/lightweightm2m-1-1-preview-3

Questions?

Thank you


OPEN SOURCE
FOUNDRIES


Embedded Linux
Conference


OpenIoTSummit