Zephyr™ Project
Developer Summit
June 8-10, 2021 ▪ @ZephyrIoT

Simplifying sensor management using LwM2M and Zephyr API

Kamil Panek
Mieszko Mieruński

Agenda

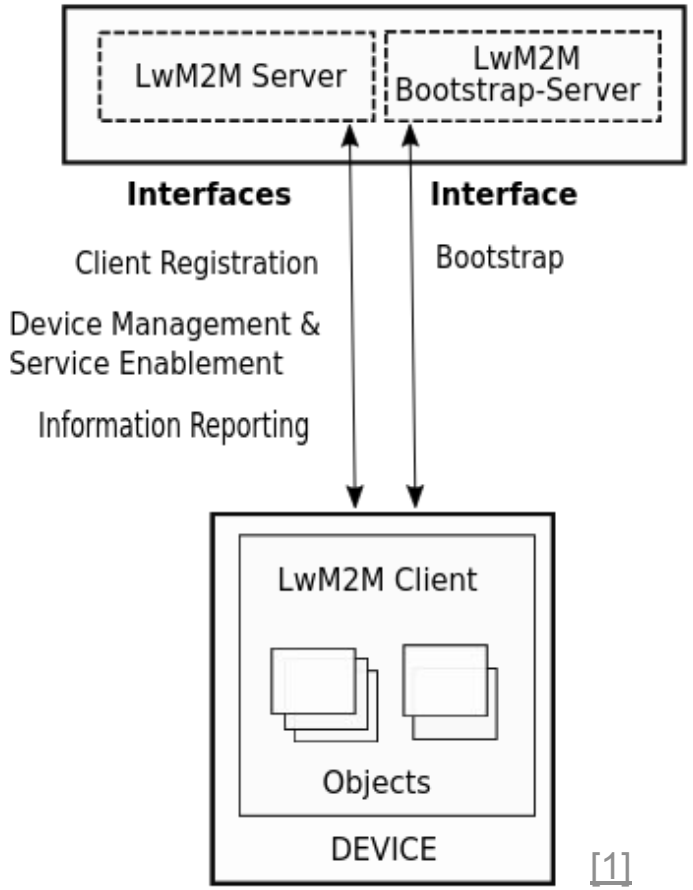
- What is LwM2M?
  - LwM2M interfaces
  - LwM2M data model
- What are IPSO objects?
- Zephyr sensor API
- Architecture of generic IPSO objects
- Application using Zephyr, Anjay LwM2M library and IPSO objects
- Sensor object implementation using Zephyr API
- Data monitoring using Coiote IoT Device Management
- Live Demo
- Questions

What is LwM2M?

- Device management protocol designed for sensor networks and the demands of M2M and IoT environment
- Messaging layer is based on CoAP (RFC 7252)
- Request/response model is used for communication between server and client
- Four interfaces are designed for communication with bootstrap and management servers

LwM2M interfaces

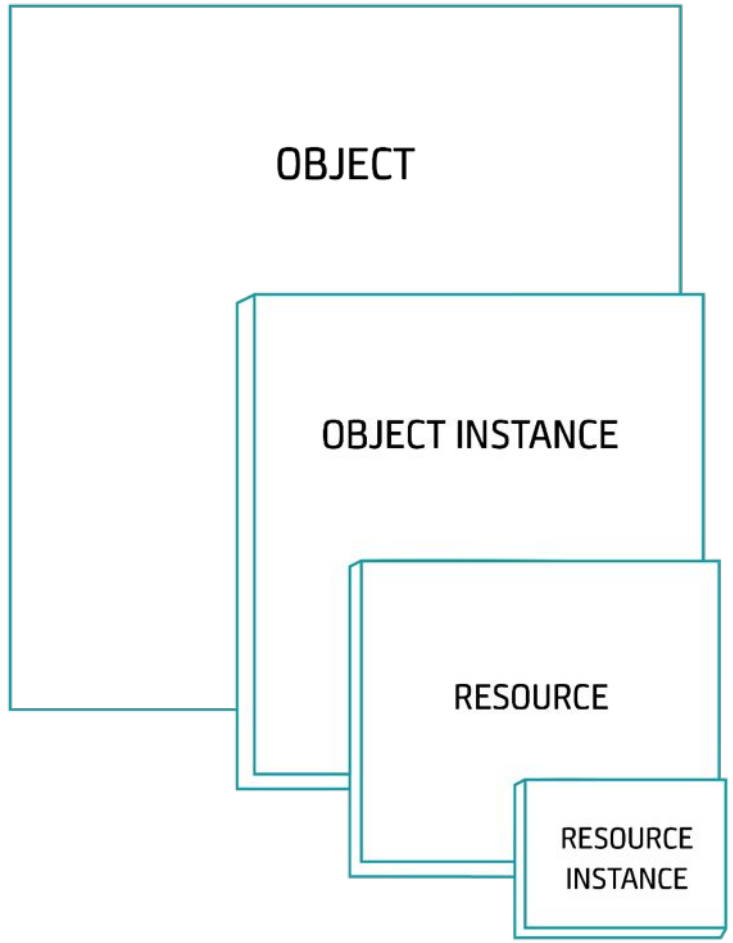
BOOTSTRAP	• Bootstrap-Request • Write, Read, Discover, Delete, Bootstrap-Finish
CLIENT REGISTRATION	• Register, De-register, Update
DEVICE MANAGEMENT & SERVICE ENABLEMENT	• **Read**, **Write**, **Execute**, Create, Delete, Write-Attributes, Discover, Read-Composite, Write-Composite, Send
INFORMATION REPORTING	• **Observe**, Observe-Composite, Cancel Observation, Cancel Observation-Composite • **Notify**



[1]

LwM2M data model

- Simple tree with maximum depth of 4
- Objects can have Object Instances
- Resources can have Resource Instances
- Resources defined into certain data types: string, integer, float, boolean, opaque and others



What are IPSO objects?

- A set of LwM2M objects mainly related to measurements of physical values
- They use shared LwM2M resource identification (RID) for the same type of value.
- Publicly available in OMA LwM2M Object and Resource Registry.
- Commonly supported by LwM2M Servers, simplifying creating complex solutions using LwM2M as a management protocol.

What are IPSO objects?

- As an example, we'll take a look at parts of the *Temperature* (OID 3303) and the *Humidity* (OID 3304) objects.

ID	Operations	Name	Instances	Mandatory	Type
5700	R	Sensor Value	Single	Mandatory	Float
5701	R	Sensor Units	Single	Optional	Float
5601	R	Min Measured Value	Single	Optional	Float
5602	R	Max Measured Value	Single	Optional	Float
5605	E	Reset Min and Max Measured Values	Single	Optional	

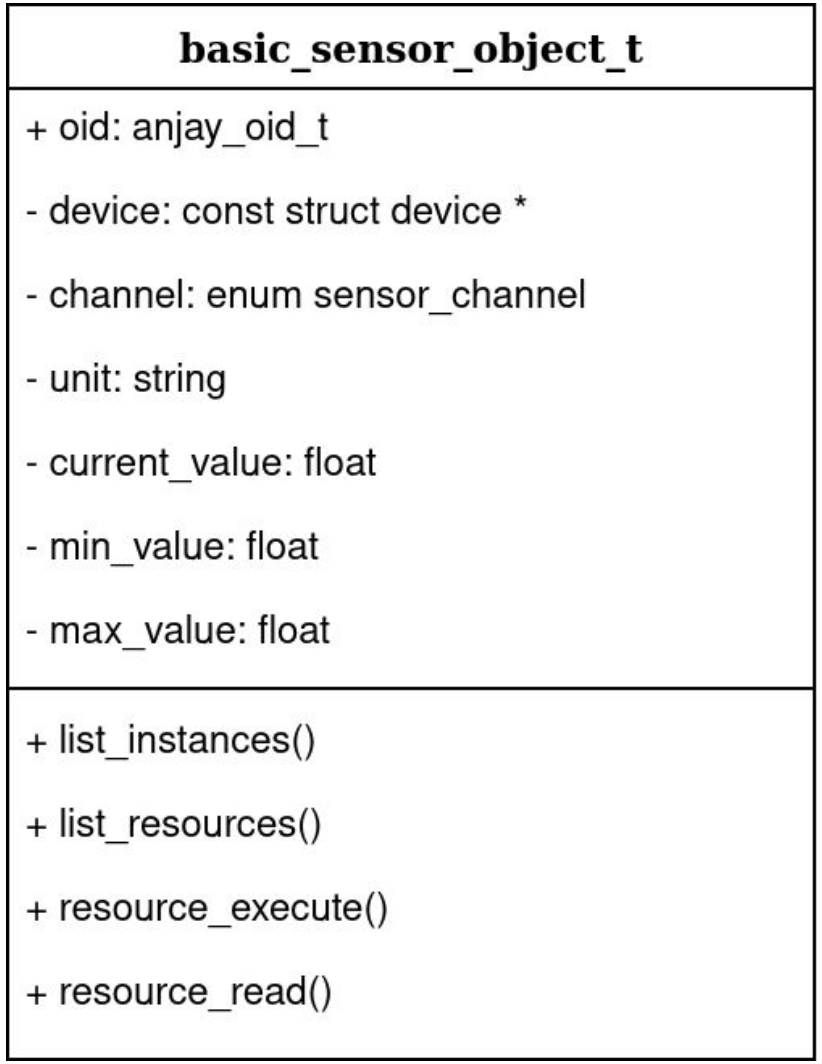
Zephyr sensor API

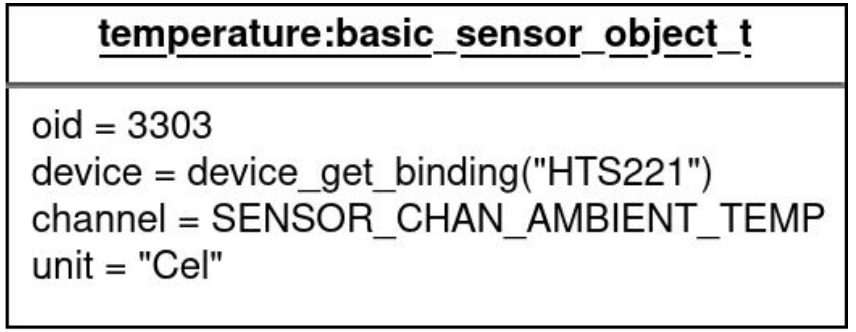
- In Zephyr, there's a universal API for various types of sensors, hiding the implementation details from user.

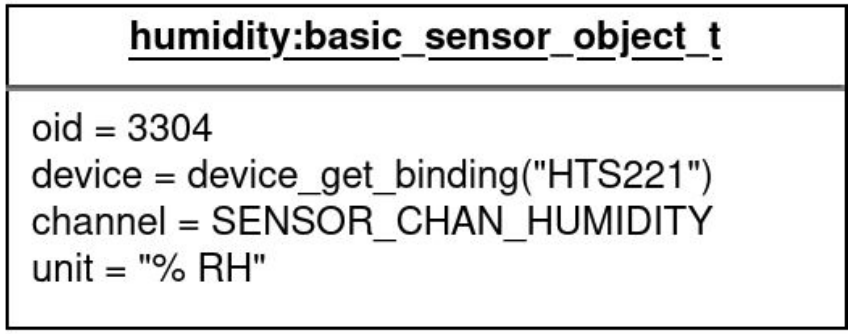
- In the simplest usage, three functions are enough to initialize the sensor and read a value from it:
  - device_get_binding()
  - sensor_sample_fetch_chan()
  - sensor_channel_get()

- To take advantage of it, we can create a "generic" IPSO object for similar types of sensors, which will be configurable during initialization.

How to implement it?







Zephyr™ Project
Developer Summit

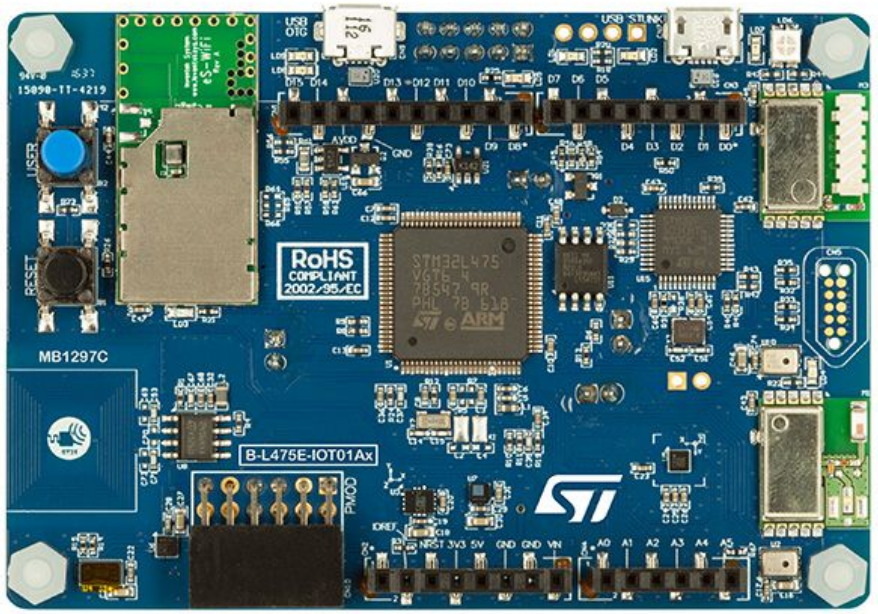
Basic client application

Environment sensor device

Zephyr™ Project

Anjay LwM2M SDK

AVSYSTEM

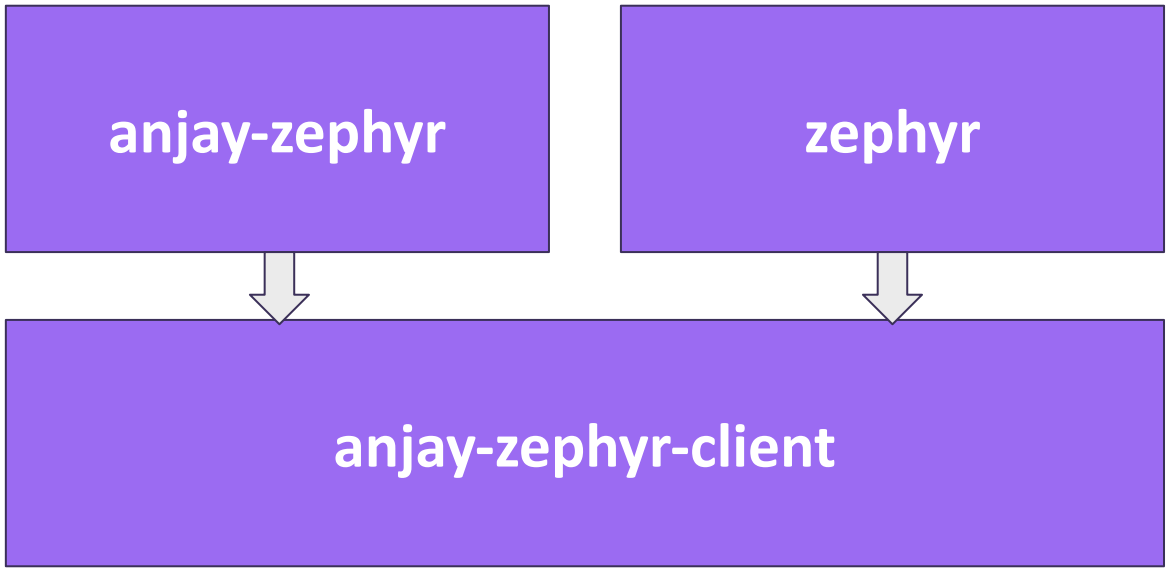


STM32 L4

[2]

Integrating Anjay LwM2M module


```
manifest:
  remotes:
    - name: zephyrproject-rtos
      url-base: https://github.com/zephyrproject-rtos
    - name: AVSystem
      url-base: https://github.com/AVSystem
  projects:
    - name: zephyr
      remote: zephyrproject-rtos
      revision: zephyr-v2.5.0
      import: true
    - name: Anjay-zephyr
      submodules: true
      remote: AVSystem
      revision: v1.0.0
      path: modules/lib/anjay
  self:
     path: Anjay-zephyr-client
```



Basic Sensor Object - Definition

- LwM2M interface handlers
- Zephyr sensor device
- Resource values

```
#define BASIC_SENSOR_OBJ_DEF(Oid)                                      \
    (anjay_dm_object_def_t) {                                          \
        .oid = (Oid),                                                  \
        .handlers = {                                                  \
            .list_instances = anjay_dm_list_instances_SINGLE,          \
            .list_resources = basic_sensor_list_resources,             \
            .resource_read = basic_sensor_resource_read,               \
            .resource_execute = basic_sensor_resource_execute          \
        }                                                              \
    }

typedef struct basic_sensor_object_struct {
    const anjay_dm_object_def_t *def_ptr;
    anjay_dm_object_def_t def;

    struct device *dev;
    enum sensor_channel channel;

    float curr_value;
    float min_value;
    float max_value;
    char unit[8];
} basic_sensor_object_t;
```

Basic Sensor Object - Create

- Get sensor device binding
- Allocate LwM2M object
- Assign starting values and parameters

```
const anjay_dm_object_def_t **
basic_sensor_object_create(const char *name,
                           enum sensor_channel channel,
                           const char *unit,
                           anjay_oid_t oid) {
    struct device *dev = device_get_binding(name);
    float value;
    if (get_value(dev, channel, &value)) {return NULL; }

    basic_sensor_object_t *obj =
            (basic_sensor_object_t *) avs_calloc(1, sizeof(basic_sensor_object_t));
    if (!obj) { return NULL; }
    obj->def = BASIC_SENSOR_OBJ_DEF(oid);
    obj->def_ptr = &obj->def;
    obj->dev = dev;
    obj->channel = channel;
    obj->curr_value = value;
    obj->min_value = value;
    obj->max_value = value;
    strcpy(obj->unit, unit);

    return &obj->def_ptr;
}
```

Basic Sensor Object - Updating Data

- Fetching data from sensors
- Notify server about changes (if requested)

```
static int
get_value(struct device *dev, enum sensor_channel channel, float *out_value)
{
    struct sensor_value value;
    if (sensor_sample_fetch_chan(dev, channel)
            || sensor_channel_get(dev, channel, &value)) {
        avs_log(sensor, ERROR, "Failed to read from %s", dev->name);
        return -1;
    }

    *out_value = (float) sensor_value_to_double(&value);
    return 0;
 }
```

```
void basic_sensor_object_update(anjay_t *anjay, const anjay_dm_object_def_t *const *def) {
    basic_sensor_object_t *obj = get_obj(def);

    float value;
    if (get_value(obj->dev, obj->channel, &value) || value == obj->curr_value) {
        return;
    }
    obj->curr_value = value;
    (void) anjay_notify_changed(anjay, obj->def.oid,0, RID_SENSOR_VALUE);

    const float min_value = AVS_MIN(obj->min_value, obj->curr_value);
    const float max_value = AVS_MAX(obj->max_value, obj->curr_value);
    if (min_value != obj->min_value) {
        (void) anjay_notify_changed(anjay, obj->def.oid,0, RID_MIN_MEASURED_VALUE);
        obj->min_value = min_value;
    }
    if (max_value != obj->max_value) {
        (void) anjay_notify_changed(anjay, obj->def.oid,0, RID_MAX_MEASURED_VALUE);
        obj->max_value = max_value;
    }
}
```

Basic Sensor Object - Resource list

- Resources present in the object

```
static int
basic_sensor_list_resources(anjay_t *anjay,
                            anjay_dm_object_def_t *const *obj_ptr,
                            anjay_iid_t iid,
                            anjay_dm_resource_list_ctx_t *ctx) {

    anjay_dm_emit_res(ctx, RID_MIN_MEASURED_VALUE, ANJAY_DM_RES_R,
                      ANJAY_DM_RES_PRESENT);
    anjay_dm_emit_res(ctx, RID_MAX_MEASURED_VALUE, ANJAY_DM_RES_R,
                      ANJAY_DM_RES_PRESENT);
    anjay_dm_emit_res(ctx, RID_RESET_MIN_AND_MAX_MEASURED_VALUES,
                      ANJAY_DM_RES_E, ANJAY_DM_RES_PRESENT);
    anjay_dm_emit_res(ctx, RID_SENSOR_VALUE, ANJAY_DM_RES_R,
                      ANJAY_DM_RES_PRESENT);
    anjay_dm_emit_res(ctx, RID_SENSOR_UNITS, ANJAY_DM_RES_R,
                      ANJAY_DM_RES_PRESENT);
    return 0;
}
```

Basic Sensor Object - Resource read

- Return resource in format specified by the IPSO registry

```
static int
basic_sensor_resource_read(anjay_t *anjay,
                           const anjay_dm_object_def_t *const *obj_ptr,
                           anjay_iid_t iid,
                           anjay_rid_t rid,
                           anjay_riid_t riid,
                           anjay_output_ctx_t *ctx) {
    basic_sensor_object_t *obj = get_obj(obj_ptr);

    switch (rid) {
    case RID_MIN_MEASURED_VALUE:
        return anjay_ret_float(ctx, obj->min_value);
    case RID_MAX_MEASURED_VALUE:
        return anjay_ret_float(ctx, obj->max_value);
    case RID_SENSOR_VALUE:
        return anjay_ret_float(ctx, obj->curr_value);
    case RID_SENSOR_UNITS:
        return anjay_ret_string(ctx, obj->unit);
    default:
        return ANJAY_ERR_METHOD_NOT_ALLOWED;
    }
}
```

Basic Sensor Object - Resource execute

- Perform execute operation on given resource

```
static int
basic_sensor_resource_execute(anjay_t *anjay,
                              anjay_dm_object_def_t *const *obj_ptr,
                              anjay_iid_t iid,
                              anjay_rid_t rid,
                              anjay_execute_ctx_t *arg_ctx) {
    basic_sensor_object_t *obj = get_obj(obj_ptr);

    switch (rid) {
    case RID_RESET_MIN_AND_MAX_MEASURED_VALUES:
        obj->max_value = obj->curr_value;
        obj->min_value = obj->curr_value;
        return 0;
    default:
        return ANJAY_ERR_METHOD_NOT_ALLOWED;
    }
}
```

Basic Sensor Object

- Multiple objects defined using single code base
  - Barometer
  - Distance
  - Temperature
  - Humidity

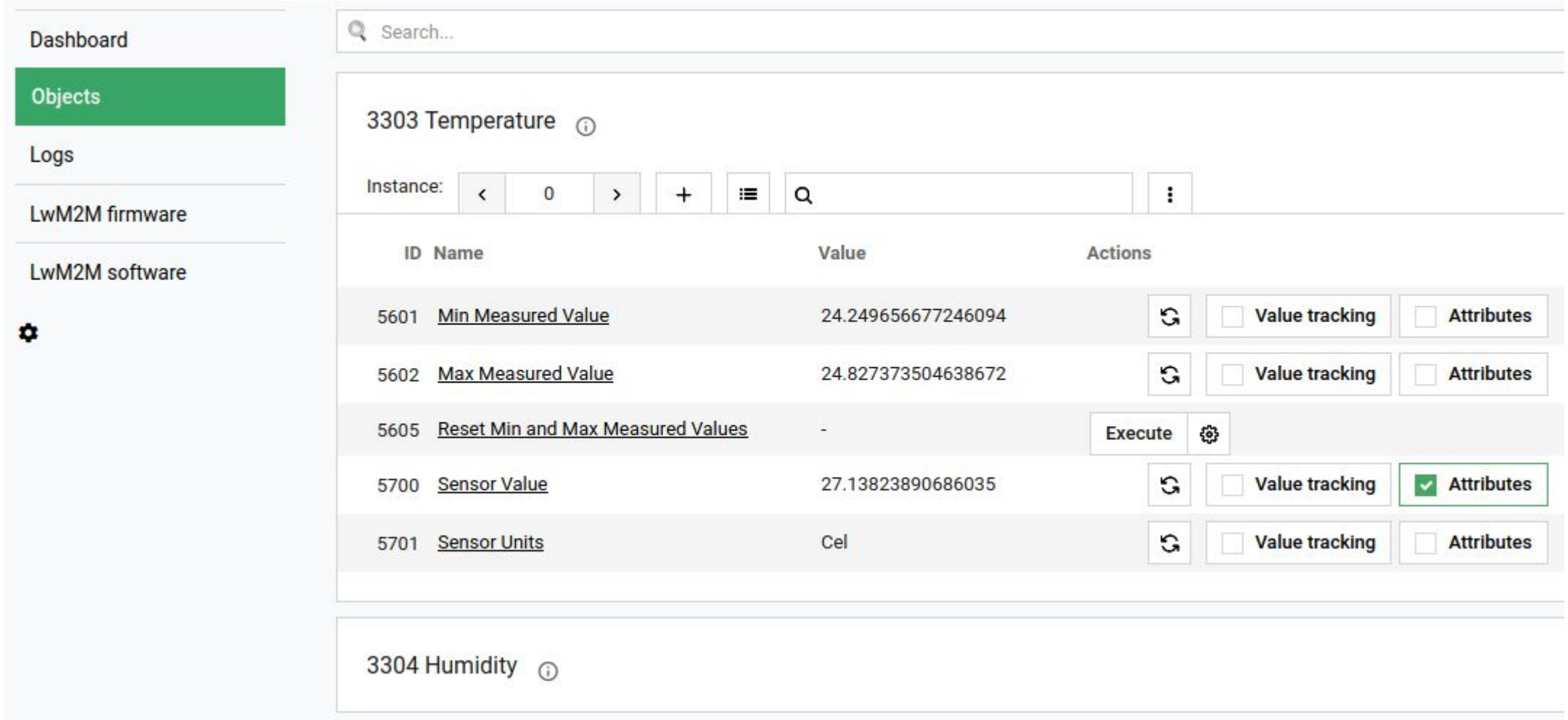
```
const anjay_dm_object_def_t **barometer_object_create(void) {
    return basic_sensor_object_create(DT_LABEL(DT_INST(0, st_lps22hb_press)),
                                      SENSOR_CHAN_PRESS,"kPa", 3315);
}

const anjay_dm_object_def_t **distance_object_create(void) {
    return basic_sensor_object_create(
        DT_LABEL(DT_INST(0, st_vl53l0x)), SENSOR_CHAN_DISTANCE,"m", 3330);
}

const anjay_dm_object_def_t **temperature_object_create(void) {
    return basic_sensor_object_create(
            "HTS221", SENSOR_CHAN_AMBIENT_TEMP,"Cel", 3303);
}

const anjay_dm_object_def_t **humidity_object_create(void) {
    return basic_sensor_object_create(
            "HTS221", SENSOR_CHAN_HUMIDITY,"% RH", 3304);
}
```

Client Application

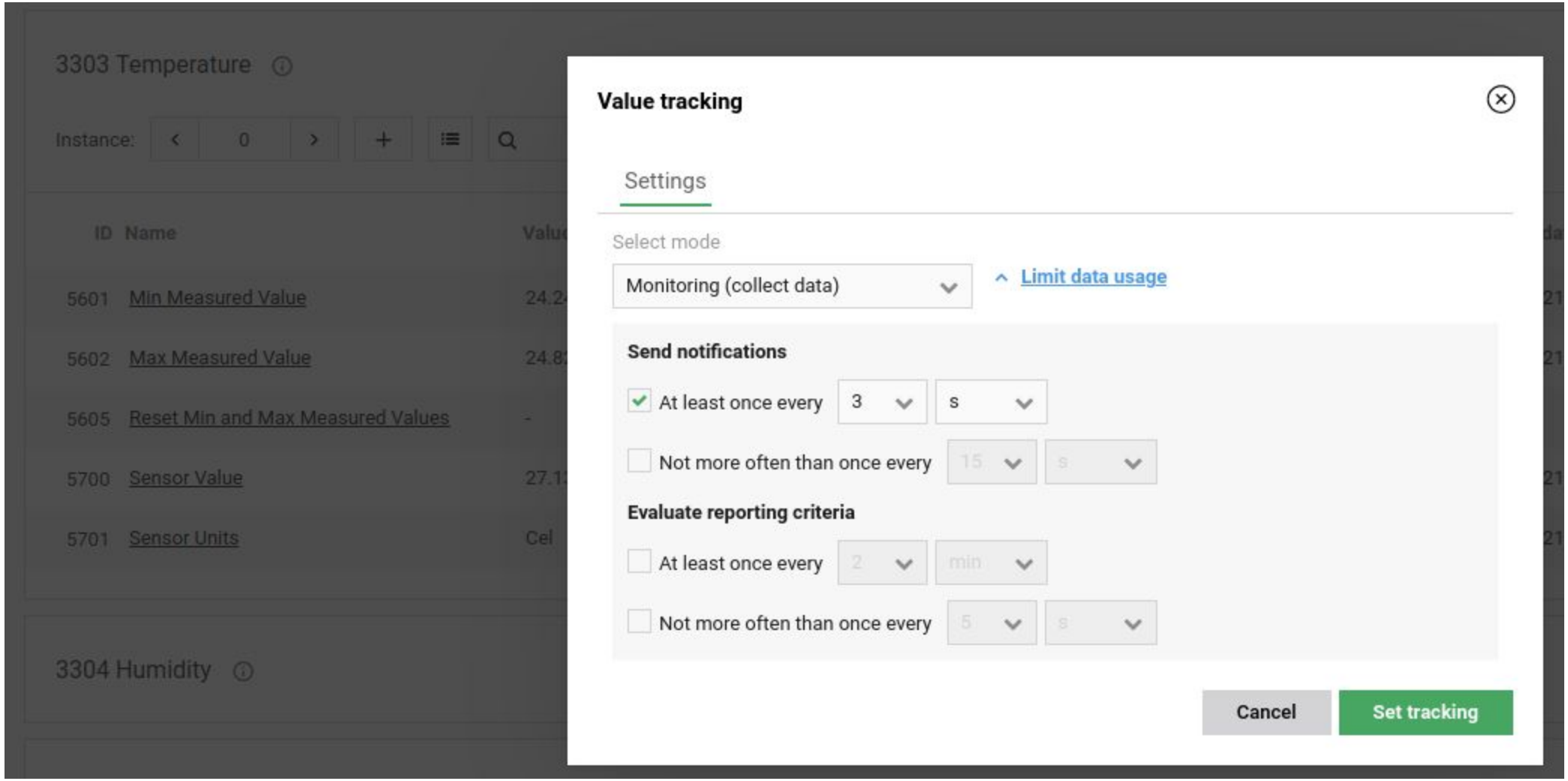
- Create new Anjay client
- Create and register sensor objects
- Install necessary objects
- The code in example is available on github (Anjay-zephyr-client)

```
anjay_t * anjay = anjay_new(&config);
anjay_dm_object_def_t **temperature_obj = temperature_object_create();
anjay_dm_object_def_t **humidity_obj = humidity_object_create();
anjay_dm_object_def_t **distance_obj = distance_object_create();
anjay_dm_object_def_t **barometer_obj = barometer_object_create();
anjay_register_object(anjay, temperature_obj);
anjay_register_object(anjay, humidity_obj);
anjay_register_object(anjay, distance_obj);
anjay_register_object(anjay, barometer_obj);
```

Data Monitoring

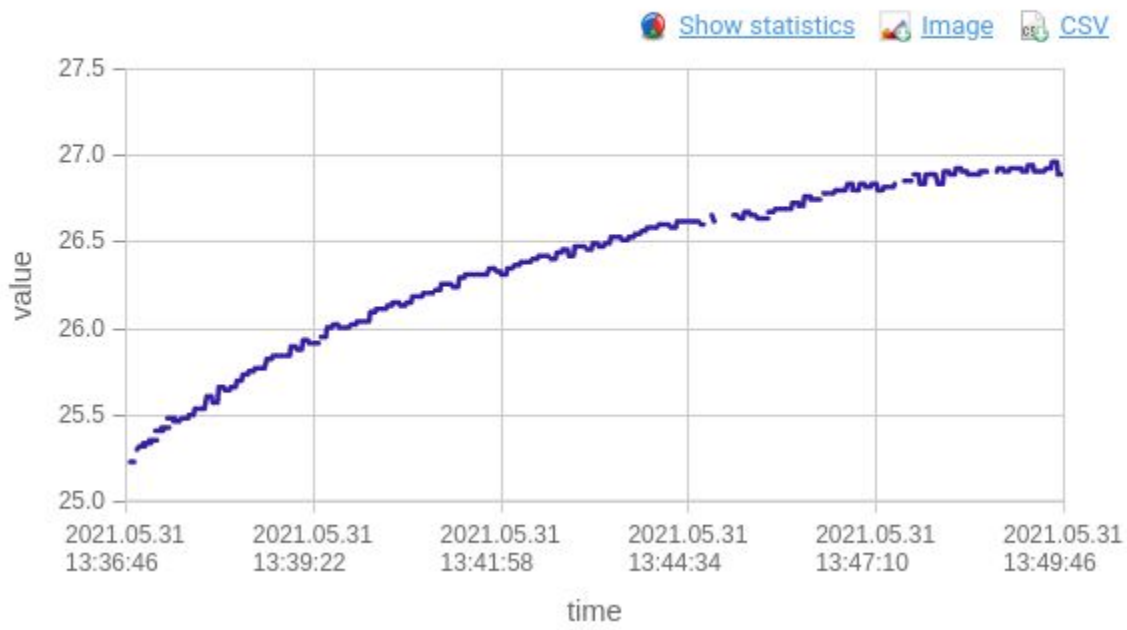


Data Monitoring



Data Monitoring

Temperature.0.Sensor Value



Humidity.0.Sensor Value

Zephyr™ Project
Developer Summit

Live Demo

Summary

- IPSO objects allows to create LwM2M-based devices compatible with servers from various vendors
- Zephyr provides a universal API for all types of sensors, which simplifies implementation of IPSO objects
- The presented approach allows to save both space on the flash memory and the time required to add new objects

Resources

- **Anjay Zephyr Client Samples**
  github.com/AVSystem/Anjay-zephyr-client
- **Anjay Zephyr Module**
  github.com/AVSystem/Anjay-zephyr
- **Coiote IoT Device Management Platform**
  https://www.avsystem.com/products/coiote-iot-device-management-platform/
- **Demo of Coiote IoT DM Platform**
  https://www.avsystem.com/try-anjay/
- Sources:
  - [1]http://www.openmobilealliance.org/release/LightweightM2M/V1_1_1-20190617-A/HTML-Version/OMA-TS-LightweightM2M_Core-V1_1_1-20190617-A.html#Figure-4-1-The-overall-architecture-of-the-LwM2M-Enabler
  - [2] https://docs.zephyrproject.org/latest/boards/arm/disco_l475_iot1/doc/index.html

Zephyr™ Project
Developer Summit

Thank you!

Zephyr™ Project
Developer Summit

Questions

Zephyr™ Project

Developer Summit

June 8-10, 2021 ▪ @ZephyrIoT