

---

# **Thunder 360/80**

## **Zigbee interface communication guide**

---

### **Introduction**

The Zigbee module enable sensor's data delivery to Zigbee network. The module supports connection to host system, updates occupancy status and motion and respiration data when detection occurs in sensor and allows to the host system to read temperature of the device and power voltage level.

Zigbee interface supports three sending mode: by event, fixed time interval and by request. The default and recommended mode is "event driven". In the event mode sensor sends the data immediately after detection and sends zero data one time per minute if there was no detection. This mode is most robust and real time option.

In the "fixed interval" mode sensor sends data with fixed interval despite if there was any detection or not.

In the "by request" mode sensor does not send any data until host system request.

Zigbee module configured as an "end device" and has 5 clusters: genBasic (0), genPowerCfg (1), genDeviceTempCfg (2), genIdentify (3), msOccupancySensing (1030).

For detailed information about the radar sensor please check the "Thunder Series Guidebook".

### **Contents**

1. Connection to Zigbee network and available clusters.....	2
1.1 Sensor's Zigbee clusters.....	3
1.1.1 msOccupancySensing (1030) .....	3
1.1.2 genPowerCfg (1) .....	7
1.1.3 genDeviceTempCfg (2).....	7
1.1.4 genBasic (0) and genIdentify (3) .....	7
1.2 ioBroker additions .....	8
2. Adding sensor in ioBroker.....	11

---

## 1. Connection to Zigbee network and available clusters

When sensor is turn on it is ready for pairing process with the host. The sensor is continuously announcing itself until it is connected to the network. If sensor is disconnected from the network, it starts to announce itself again.

It is configured as end device and has 5 clusters: genBasic (0), genPowerCfg (1), genDeviceTempCfg (2), genIdentify (3), msOccupancySensing (1030).

To work in zigbee2mqtt and in ioBroker.Zigbee the description of the device should be added in the zigbee-herdsman-converters/devices.js as shown below:

Table 1 – Addition to zigbee-herdsman-converters/devices.js

```
{  
  zigbeeModel: ['Umain_Thunder'],  
  model: 'Umain_Thunder',  
  vendor: 'Umain',  
  description: 'Occupancy sensor with motion and respiration detection features',  
  supports: 'Occupancy (motion, respiration with BPM) and device temperature and battery voltage (cc2530 internal measures)',  
  fromZigbee: [fz.device_temperature, fz.battery, fz.umain_occupancy],  
  toZigbee: [tz.ultrasonicUToOThreshold, tz.ultrasonicUToODelay, tz.ultrasonicOToUDelay],  
}
```

The file zigbee-herdsman-converters/converters/fromZigbee.js is also should be updated. The code is provided in cluster description ([1.1.1 msOccupancySensing \(1030\)](#)).

---

## 1.1 Sensor's Zigbee clusters

### 1.1.1 msOccupancySensing (1030)

The cluster implementation supports four attributes:

- 1) 0x0000 **Occupancy** map8 0b0000 000x RP – M

The **Occupancy** attribute is a bitmap.

Bit 0 specifies the sensed occupancy as follows: 1 = occupied, 0 = unoccupied.

According to standard all other bits are reserved. In case of Thunder radar sensor, the other bits contain data about motion and respiration detections.

The number over 20 and below 1000 indicates that motion is detected.

The number over 1000 indicates that respiration is detected and represent instant estimation of BPM value.

Example shows how it can be extracted and interpreted:

Take the value of the attribute. The motion value can be taken as a division remainder of the input attribute value by 1000 and check if it is more than 20:

$$\text{motion} = (\text{msg.data.occupancy \% 1000}) >= 20 ? (\text{msg.data.occupancy \% 1000}) : 0$$

The BPM value of detected respiration can be extracted and an integer result of division by 1000:

$$\text{Respiration\_bpm} = \text{Math.floor}(\text{msg.data.occupancy}/1000)$$

- 2) 0x0022 **UltrasonicUnoccupiedToOccupiedThreshold** uint8 0x01 – 0xfe RW 0x01 O

The **UltrasonicUnoccupiedToOccupiedThreshold** attribute is 8 bits in length and in our implementation, it is a control parameter that specifies the maximum detection distance on the radar and responsible for radar reboot function. Possible values:

**0** – reboot radar sensor

**1** – set 1.5m

**5** – set 3.5m

**2** – set 2.0m

**6** – set 4.0m

**3** – set 2.5m

**7** – set 4.5m

**4** – set 3.0m

**8** – set 5.0m

- 
- 3) *0x0021 UltrasonicUnoccupiedToOccupiedDelay uint16 0x0000 – 0xffff RW 0x00 0*

The ***UltrasonicUnoccupiedToOccupiedDelay*** attribute is 16 bits in length and in our implementation, it is a control parameter that specifies the transfer mode of the sensor's data and interval time. Possible values:

**0** – event mode - **recommended\***.

**1...180** – fixed sending interval, the value represents the interval in seconds.

**190** – data is not sending until it is requested by host.

\*in the event mode sensor sends the data immediately after detection and zero data one time per minute if there was no detection. This mode is most robust and real time option.

- 4) *0x0020 UltrasonicOccupiedToUnoccupiedDelay uint16 0x0000 – 0xffff RW 0x00 0*

The ***UltrasonicOccupiedToUnoccupiedDelay*** attribute is 16 bits in length and in our implementation this field is needed for “fixed interval” or “by request” sending mode and it contains the number of motion and respiration detections during the period since last reading. And the ***Occupancy*** data contains the average value of motion/respiration(BPM) over the period.

In the “fixed interval” mode this field must be requested by host during 2 seconds after detection data was updated. Then this field is zeroing automatically.

In the mode when data is requested by host this field must be zeroed by the host after reading.

This data is meaningful for activity estimation and respiration BPM estimation (i.e., if there was only one respiration detection during last 5 minutes the BPM value should not be considered a trustworthy).

The interaction with this radar cluster should be described in the script that works with Zigbee adapter (fromZigbee.js and toZigbee.js) as it shown in Table 2 and Table 3 accordingly:

Table 2 – Addition to zigbee-herdsman-converters/converter/fromZigbee.js

```
umain_occupancy: {
    // This is for Umain Thunder occupancy sensor that send motion and respiration detections
    cluster: 'msOccupancySensing',
    type: ['attributeReport', 'readResponse'],
    convert: (model, msg, publish, options, meta) => {
        if (msg.data.hasOwnProperty('occupancy')) {
            const occupancy_val = Math.abs(parseInt(msg.data.occupancy));
            const motion = (occupancy_val % 1000) >= 20 ? (occupancy_val % 1000) : 0;
            return {
                occupancy: (occupancy_val % 2) > 0,
                occupancy_raw: occupancy_val,
                occupancy_motion: motion,
                occupancy_respiration: Math.floor(occupancy_val/1000)
            };
        }
        if (msg.data.hasOwnProperty('ultrasonicUToOThreshold')) {
            return {ultrasonicUToOThreshold: parseInt(msg.data['ultrasonicUToOThreshold'])};
        }
        if (msg.data.hasOwnProperty('ultrasonicUToODelay')) {
            return {ultrasonicUToODelay: parseInt(msg.data['ultrasonicUToODelay'])};
        }
        if (msg.data.hasOwnProperty('ultrasonicOToUDelay')) {
            return {NumOfMotions: Math.floor(parseInt(msg.data['ultrasonicOToUDelay'])/256),
                    NumOfRespirations: (parseInt(msg.data['ultrasonicOToUDelay'])%256),
                    ultrasonicOToUDelay: (parseInt(msg.data['ultrasonicOToUDelay']))};
        }
    },
},
```

Table 3 – Addition to zigbee-herdsman-converters/converterstoZigbee.js

```
ultrasonicUToOThreshold: {
    // set threshold for motion detector; for radar sensor this sets maximum working distance
    key: ['ultrasonicUToOThreshold'],
    convertSet: async (entity, key, value, meta) => {
        value *= 1;
        await entity.write('msOccupancySensing', {ultrasonicUToOThreshold: value});
        return {state: {ultrasonicUToOThreshold: value}};
    },
    convertGet: async (entity, key, meta) => {
        await entity.read('msOccupancySensing', ['ultrasonicUToOThreshold']);
    },
},
ultrasonicUToODelay: {
    // set delay after motion detector changes from unoccupied to occupied; for radar sensor this sets
    interval sending parameter
    key: ['ultrasonicUToODelay'],
    convertSet: async (entity, key, value, meta) => {
        value *= 1;
        await entity.write('msOccupancySensing', {ultrasonicUToODelay: value});
        return {state: {ultrasonicUToODelay: value}};
    },
    convertGet: async (entity, key, meta) => {
        await entity.read('msOccupancySensing', ['ultrasonicUToODelay']);
    },
},
ultrasonicOToUDelay: {
    // set delay after motion detector changes from occupied to unoccupied; for radar sensor this stores
    number of detections since last reading
    key: ['ultrasonicOToUDelay'],
    convertSet: async (entity, key, value, meta) => {
        value *= 1;
        await entity.write('msOccupancySensing', {ultrasonicOToUDelay: value});
        return {state: {ultrasonicOToUDelay: value}};
    },
    convertGet: async (entity, key, meta) => {
        await entity.read('msOccupancySensing', ['ultrasonicOToUDelay']);
    },
},
```

---

### 1.1.2 genPowerCfg (1)

The cluster implementation supports one attributes:

0x0020 **BatteryVoltage** uint8 0x00 – 0xff Read Only – O

The attribute can be read to control power voltage level. The converter for this cluster is standard.

### 1.1.3 genDeviceTempCfg (2)

The cluster implementation supports one attributes:

0x0000 **CurrentTemperature** int16 -200 to +200 Read Only - M

The attribute can be read to control temperature of the sensor module. The converter for this cluster is standard.

### 1.1.4 genBasic (0) and genIdentify (3)

The clusters are supporting the minimum standard attributes.

---

## 1.2 ioBroker additions

The device and convertor descriptions provided above are enough to connect the sensor to zigbee2mqtt service. For the ioBroker platform need to make few more updates:

Table 4 – Addition to ioBroker.zigbee/lib/devices.js

```
{  
  models: ['Umain_Thunder'],  
  icon: 'img/thunder.png',  
  states: [  
    states.device_temperature,  
    states.voltage,  
    states.occupancy,  
    states.occupancy_raw,  
    states.occupancy_motion,  
    states.occupancy_respiration,  
    states.NumOfMotions,  
    states.NumOfRespirations,  
    states.ultrasonicUToOThreshold,  
    states.ultrasonicUToODelay,  
    states.ultrasonicOToUDelay,  
  ],  
}
```

Table 5 – Addition to ioBroker.zigbee/lib/states.js

```
voltage: {  
  id: 'voltage',  
  name: 'Battery voltage',  
  icon: 'img/battery_v.png',  
  role: 'battery.voltage',  
  write: false,  
  read: true,  
  type: 'number',  
  unit: 'V',  
  getter: payload => payload.voltage / 1000,  
},  
device_temperature: {  
  id: 'device_temperature',  
  name: 'Device Temperature',  
  icon: undefined,  
  role: 'value.temperature',  
  write: false,  
  read: true,  
  type: 'number',  
  unit: "C"  
},
```

```

occupancy_raw: {
    id: 'occupancy_raw',
    name: 'Occupancy raw',
    icon: undefined,
    role: 'sensor.motion',
    write: false,
    read: true,
    type: 'number',
},
occupancy_motion: {
    id: 'occupancy_motion',
    name: 'Occupancy motion',
    icon: undefined,
    role: 'sensor.motion',
    write: false,
    read: true,
    type: 'number',
},
occupancy_respiration: {
    id: 'occupancy_respiration',
    name: 'Occupancy respiration',
    icon: undefined,
    role: 'sensor.motion',
    write: false,
    read: true,
    type: 'number',
},
occupancy: {
    id: 'occupancy',
    name: 'Occupancy',
    icon: undefined,
    role: 'sensor.motion',
    write: false,
    read: true,
    type: 'boolean',
},
ultrasonicUToOThreshold: {
    //this is different from occupancy_timeout,
    //is writable threshold (to device).
    id: 'ultrasonicUToOThreshold',
    prop: 'ultrasonicUToOThreshold',
    name: 'Threshold U to O; Distance for radar sensor',
    icon: undefined,
    role: 'value.ultrasonicUToOThreshold',
    write: true,
    read: true,
    type: 'number',
}

```

```

ultrasonicUToODelay: {
    id: 'ultrasonicUToODelay',
    prop: 'ultrasonicUToODelay',
    name: 'Delay U to O; Sending interval parameter for radar sensor',
    icon: undefined,
    role: 'value.ultrasonicUToODelay',
    write: true,
    read: true,
    type: 'number',
},
ultrasonicOToUDelay: {
    id: 'ultrasonicOToUDelay',
    prop: 'ultrasonicOToUDelay',
    name: 'Delay O to U; Number of detections since last reading for radar sensor',
    icon: undefined,
    role: 'value.ultrasonicOToUDelay',
    write: true,
    read: true,
    type: 'number',
},
NumOfMotions: {
    id: 'NumOfMotions',
    prop: 'NumOfMotions',
    name: 'Number of motion detections since last reading for radar sensor',
    icon: undefined,
    role: 'value.NumOfMotions',
    write: true,
    read: true,
    type: 'number',
},
NumOfRespirations: {
    id: 'NumOfRespirations',
    prop: 'NumOfRespirations',
    name: 'Number of respiration detections since last reading for radar sensor',
    icon: undefined,
    role: 'value.NumOfRespirations',
    write: true,
    read: true,
    type: 'number',
},

```

The icon (`icon: 'img/thunder.png'`) should be available on the folder '`img/thunder.png`' and can be downloaded from [umain.co.kr](http://umain.co.kr) website.

## 2. Adding sensor in ioBroker

After all implementation of device and convertors description described above the sensor can be connected to the Zigbee adapter of ioBroker platform. Provide to power to the sensor and run “Pairing process” and wait until sensor announce itself.

After that you can find connected sensor in the system:

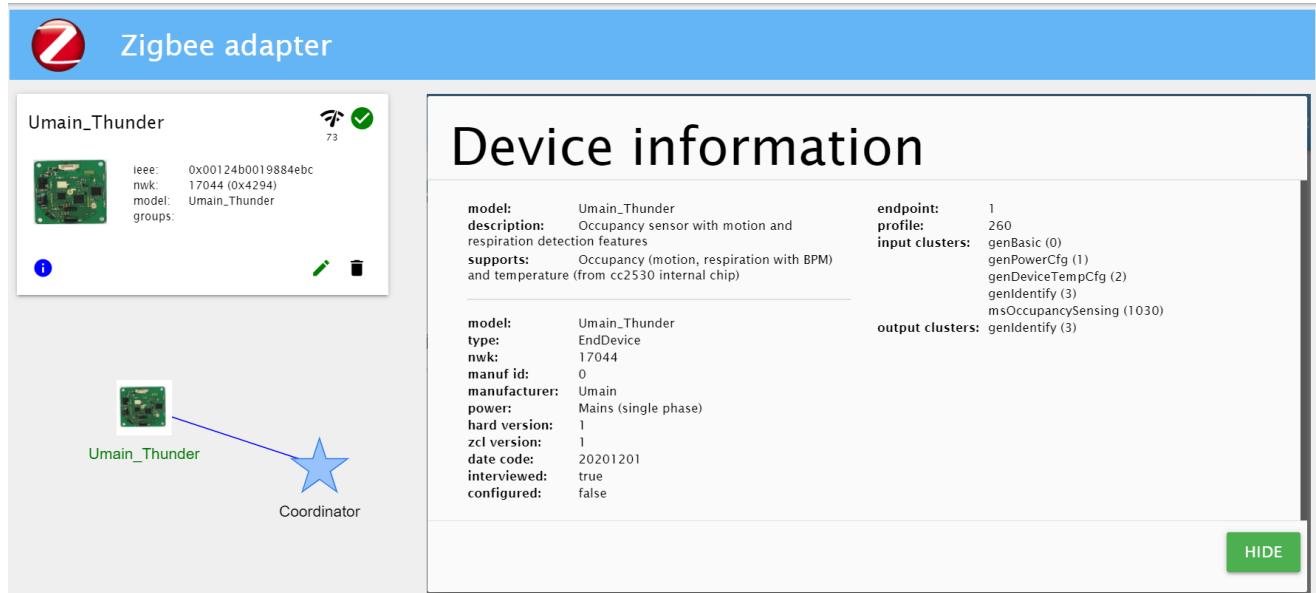


Fig. 1 – Thunder occupancy sensor registered in the ioBroker Zigbee adapter

On the “Objects” tab of ioBroker it will show the sensor’s settings and real time data updates:

zigbee.0					
00124b0019884ebc	Umain_Thunder	device			
NumOfMotions	Number of motion detections since last reading for radar s...	state	value.NumOfMotions		
NumOfRespirations	Number of respiration detections since last reading for rad...	state	value.NumOfRespirations		
available	Available	state	state	true	
device_temperature	Device Temperature	state	value.temperature	25.28 °C	
link_quality	Link quality	state	state	118	
occupancy	Occupancy	state	sensor.motion	true	
occupancy_motion	Occupancy motion	state	sensor.motion	45	
occupancy_raw	Occupancy raw	state	sensor.motion	45	
occupancy_respiration	Occupancy respiration	state	sensor.motion	0	
ultrasonicOToUDelay	Delay O to U; Number of detections since last reading for r...	state	value.ultrasonicOToUDelay	0	
ultrasonicUToODelay	Delay U to O; Sending interval parameter for radar sensor	state	value.ultrasonicUToODelay	0	
ultrasonicUToOThreshold	Threshold U to O; Distance for radar sensor	state	value.ultrasonicUToOThreshold	6	
voltage	Battery voltage	state	battery.voltage	3.2 V	

Fig. 2 – Thunder occupancy sensor attributes in “Objects” tab

In the default mode the occupancy data updated by sensor automatically when sensor detects motion. But in the “fixed interval” mode the data updated with fixed interval and the number of detections should be requested by host. The parameters of device temperature and voltage also can be read by request. The example of script to do this is show in Table 6.

The parameters of sensor responsible for working distance and sending mode can be changed using this form (fig. 2). If to set working distance to 0 value then sensor will reboot. To set it back to actual distance need to read this parameter from the sensor. The script to do this is shown in Table 6 as well.

Table 6 – JavaScript for ioBroker to update parameters automatically.

```
/*
This script is an example of how to manage data of the Umain Thunder sensor
*/

/*----- New sensors can be added here -----*/
var id_set = [ 'zigbee.0.00124b0019884ebc'/*Kitchen sensor*/,
               'zigbee.0.00124b0019884e82'/*Bedroom sensor*/
];
/*-----*/
var interval4, interval3, interval, interval2;

/*      cid      zclData    */
var battery_addr = ["genPowerCfg", "batteryVoltage"];
var temerature_addr = ["genDeviceTempCfg", "currentTemperature"];
var interval_time_addr = ["msOccupancySensing", "ultrasonicUToODelay"];
var num_of_alarms_addr = ["msOccupancySensing", "ultrasonicOToUDelay"];
var distance_addr = ["msOccupancySensing", "ultrasonicUToOThreshold"];

// Request sensor data
function GET_SENSOR_DATA(id_var, parameter_addr) {
  var cid_var = parameter_addr[0];
  var zcldata_var = { [parameter_addr[1]]: {} };
  try {
    sendTo('zigbee.0', 'sendToZigbee', {
      "id": id_var,
      "ep": "1",
      "cid": cid_var,
      "cmd": "read",
      "cmdType": "foundation",
      "zclData": zcldata_var,
      "cfg": null
    });
  } catch (e) { console.error(e); }
}
```

```

interval4 = setInterval(function () {
  for (var id_num in id_set)
  {
    GET_SENSOR_DATA(id_set[id_num], temerature_addr);
    console.log('Temperature of '+id_set[id_num] + String(((' ' +
    getState(id_set[id_num] + ".device_temperature").val)));
  }, 30000);

interval3 = setInterval(function () {
  for (var id_num in id_set)
  {
    GET_SENSOR_DATA(id_set[id_num], battery_addr);
    console.log('Voltage of '+id_set[id_num] + String(((' ' +
    getState(id_set[id_num] + ".voltage").val)));
  }, 50000);

interval = setInterval(function () {
  for (var id_num in id_set)
  {
    var value = getState(id_set[id_num] + ".ultrasonicUToOThreshold").val;
    if (value == null || parseFloat(value) == 0)  {
      GET_SENSOR_DATA(id_set[id_num], distance_addr);
      console.log('Distance of '+id_set[id_num] + String(((' ' + value)));
    }
  }, 1000);

/*----- If you use interval mode then set event handler for each sensor -----*/
on({id: id_set[0] + ".occupancy_raw", change: "any"}, function (obj) {
  if (getState(id_set[0] + ".ultrasonicUToODelay").val == null) {
    GET_SENSOR_DATA(id_set[0], interval_time_addr);
  }
  else if (getState(id_set[0] + ".ultrasonicUToODelay").val != 0) {
    GET_SENSOR_DATA(id_set[0], num_of_alarms_addr);
    console.log('Num of alarms ' + id_set[0] + String(((' ' + getState(id_set[0] + ".ultrasonicOToUDelay").val)));
  }
});

on({id: id_set[1] + ".occupancy_raw", change: "any"}, function (obj) {
  if (getState(id_set[1] + ".ultrasonicUToODelay").val == null) {
    GET_SENSOR_DATA(id_set[1], interval_time_addr);
  }
  else if (getState(id_set[1] + ".ultrasonicUToODelay").val != 0) {
    GET_SENSOR_DATA(id_set[1], num_of_alarms_addr);
    console.log('Num of alarms ' + id_set[1] + String(((' ' + getState(id_set[1] + ".ultrasonicOToUDelay").val)));
  }
});

```

To log and visualize the sensor data the “History” and “Flot” adapters can be used. The result can be represented as it shown below:

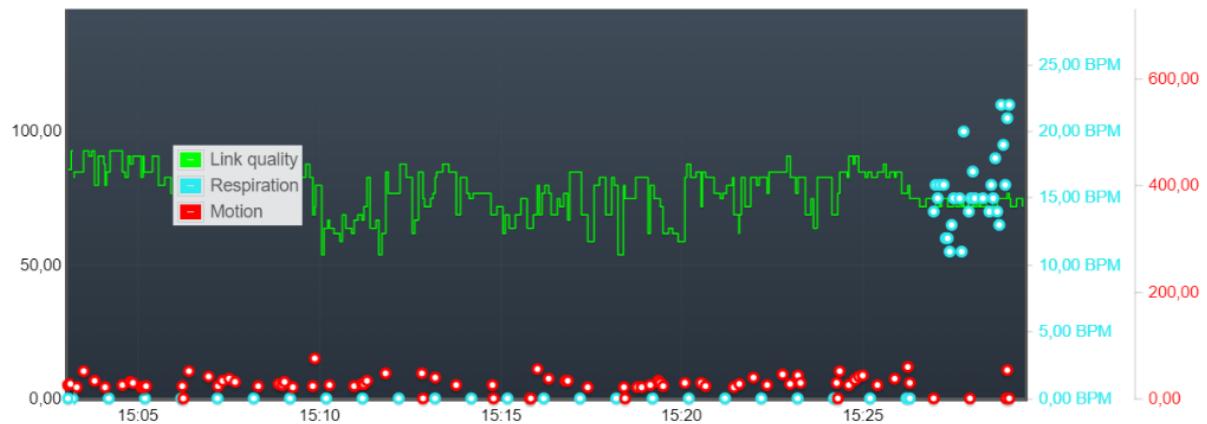


Fig. 3 – The occupancy data visualization: red – motion detection, blue – respiration detection.