The CoIoT Protocol for Shelly devices

The CoIoT protocol is yet another protocol for IoT communication and integration. CoIoT is based on CoAP with some additions as new request code 0.30 for status publishing. All payloads are JSON encoded. All responses are piggyback send with the acknowledgment to further simplify CoAP implementation.

Every CoIoT device is expected to handle a set of request URIs and generate responses in predefined format. Also every device is required to periodically send a multicast CoAP request with code 0.30 that describes it state.

Every CoAP request package and every response with payload should also carry some mandatory CoAP options (headers).

CoIoT mandatory CoAP options

Some nonstandad CoAP options are required to be transmitted to help quickly identify the remote device and determine if further processing is needed. The options are numbered with the help of this C/C++ preprocessor macros:

```
#define COIOT_OPTION_BASE 3332 //non critical, save to forward, no cache key
#define COIOT_OPTION_INCR 8
#define COIOT_OPTION_N(n) (COIOT_OPTION_BASE+(COIOT_OPTION_INCR)*(n))

#define COIOT_OPTION_GLOBAL_DEVID    COIOT_OPTION_N(0)
#define COIOT_OPTION_MGR_GLOBAL_NBASE 1
#define COIOT_OPTION_STATUS_NBASE 10
#define COIOT_OPTION_STATUS_VALIDITY COIOT_OPTION_N(COIOT_OPTION_STATUS_NBASE+0)
#define COIOT_OPTION_STATUS_SERIAL   COIOT_OPTION_N(COIOT_OPTION_STATUS_NBASE+1)
#define COIOT_OPTION_MGR_STATUS_NBASE 20
```

To follow CoAP standard, options are numbered above 2048 where proprietary options should reside. LS Bits carry some CoAP proxy flags so we guard them with spacing the numbers with increment of 8.

COIOT_OPTION_GLOBAL_DEVID

First defined option is `COIOT_OPTION_GLOBAL_DEVID` This is global option and must be present in all CoIOT packages with request or payload. It's value is string in format `<devtype>#<devid>#<version>` for example `SHSEN-1#4B3F9E#1` The whole option should be less than 50 bytes.

Other mandatory options are `COIOT_OPTION_STATUS_VALIDITY` and `COIOT_OPTION_STATUS_SERIAL` this are mandatory only in status responses or publishes.

COIOT_OPTION_STATUS_VALIDITY

`COIOT_OPTION_STATUS_VALIDITY` is uint16_t in network byte order (big endian) that states the maximal time between this and the next status publish. This way a device can state it's report interval. If a report is not received from this device after the interval has passed the device should be considered offline. The LS bit of this option controls how the values is scaled. if the LS bit is 0 the value is number of 1/10 of seconds in the validity period so 2 is 0.2 seconds, 10 is a second, 600 is a minute, 65534 is 109 minutes and 13 seconds. If the LS bit of `COIOT_OPTION_STATUS_VALIDITY` is 1 the value is number of 4 seconds interval in the whole interval. So 3 is 12 seconds 11 is 44 seconds and 65535 is more than 3 days.

COIOT_OPTION_STATUS_SERIAL

This option is mandatory in status response and publishes. It is a uint16_t in network byte order which indicates a change in the status report. When a new status report is handled all payload processing can be skipped if the serial number does not change from the last processed payload. The value `0` is reserved and should not be send. This allows easy initialization in the receiving devices.

Reserved options numbers

Devices are free to define and use their own options but we reserve some for future protocol development.

Device description (/cit/d)

Every device should response to CoAP GET request with URI /cit/d and return JSON payload describing the device. Throughout the description all `<id>` values should be non overlapping integers that are used as unique identificators of blocks and sensors

toplevel object should look like:

```
{
"blk": [....],
"sen": [....],
"act": [....]
}
```

`blk` array should hold list of all "blocks" of the device. Each device should have at least one block. For example if you're device exposes just few sensors it needs just one block, but if you have a multi channel relay you should define a block for each relay. Each sensor or action should be linked to one or many blocks to help users better understand what is measured and what is executed in more complex devices.

`blk` elements should be objects with structure

```
{"I":<id>, "D":<desctiption>}
```

`sen` array should hold list of all sensors and states in the device. These should be objects with structure:

```
{"I":<id>, "T":<type>, "D":<description>, "R":<range>, "L":<links>}
```

supported sensor types:

<type>	**type**	**mesured in**
"T"	Temperature	Degr.Celsius
"H"	Humidity	Percent
"L"	Luminosity	Lux
"M"	Motion	1:Motion detected; 0 No motion
"A"	Air quality	0 to 10 : best to worst
"P"	Power	Watt
"B"	Bettery Percentage	100:battery is fully charged; 20 and below Battery need to be changed/charged
"S"	Status (this is the catch-all type if no other fits)	

D field is mandatory giving one-word description of the sensor.

L field is mandatory array of int or single integer with id's of device blocks to which this sensor relates (you can have two relays with single power meter).

R filed is range description, optional, in text form either "from/to" value or in form `<I|U><8|16|32>` showing expected `S`igned or `U`nsigned integer size

Example sensor description :

```
{
  "blk": [
    {
      "I": 1,
      "D": "sensors"
    }
  ],
  "sen": [
    {
      "I": 11,
      "D": "motion",
      "T": "S",
      "R": "0/1",
      "L": 1
    },
    {
      "I": 22,
      "D": "charger",
```

```
      "T": "S",
      "R": "0/1",
      "L": 1
    },
    {
      "I": 33,
      "D": "temperature",
      "T": "T",
      "R": "-40/125",
      "L": 1
    },
    {
      "I": 44,
      "D": "humidity",
      "T": "H",
      "R": "0/100",
      "L": 1
    },
    {
      "I": 66,
      "D": "lux",
      "T": "L",
      "R": "0/1",
      "L": 1
    },
    {
      "I": 77,
      "D": "battery",
      "T": "B",
      "R": "0/100",
      "L": 1
    }
  ]
}
```

`act` elements will describe actions possible with the device in future extension of CoIoT protocol

Device status (/cit/s)

Every device should response to CoAP GET request with URI /cit/s and return JSON payload describing the device. Every device should periodically "publish" it's status using multicast packet in the form of non confirmable request with code 0.30 and request path /cit/s This code is non standard CoAP code and all CoAP complaint servers should silently ignore it. Throughout the status report all `<id>` values should match sensors ids from device description. The JSON payload should follow the form:

```
{
  "G":
    [
      [0,<senid>,<value>],
      [0,<senid>,<value>],
      ....
    ]
```

```
}
```

The `G` key stands for `Generic`. Currently all sensor values are generic and non encrypted. Future extensions of the protocol might add `P` for `Private` and define some encryption scheme.

First `0` in sensor values tuples stand for the channel number. All sensors are required to be "emitted" in channel 0. Future extensions of the protocol might define a way for the users to define extra "mapping" for sensors to channels that will be **added** to the status after the values from channel 0. This will allow for easy reconfiguration of peer to peer network. For example a multi zone alarm system can be configured to react based on channel number activity and not to have to explicit list every sensor on every siren. Currently the first position in the sensor value tuple is reserved and should be 0.

The second position is for the sensor id, and the third is for the sensor's current value.

Given the description of the sensor above here is the status of the same sensor:

```
{
  "G":[
    [0,11,0],
    [0,22,0],
    [0,33,24.58],
    [0,44,59.90],
    [0,66,11.68],
    [0,77,100]
  ]
}
```

The first tuple is for sensor id 11 that in description have `D:"motion"`. The second is for 22 - "charger", third for "temperature" and so on

Future plans for the CoIoT protocol

We intend to devise a scheme for device actions to be described in the device descriptor and executed latter on. We're also considering implementing mechanism for securely publishing of private sensor data intermixed with publicly available sensor data. We dream of pure peer-to-peer automation where each actuator is **flexibly** programmed to react to one or many sensors and where adding or replacing senor or actuator does not involve reconfiguration of the whole network.

Any suggestions are welcomed!

Allterco Robotics Ltd