**CODING**

**EMBEDDED C CODING**

|  |
| --- |
| #include < ESP8266WiFi.h >  #include<WiFiClient.h> #include <RelayCKT.h> |
| #include <ESP8266WebServer.h> |
| #include <ESP8266mDNS.h> |
| MDNSRespondermdns; // *Microcontroller coding via opensource* *https://xod.io/ide/* |
| // Replace with your network credentials |
| const char\* ssid = "YOUR\_WIFI\_SSID"; |
| const char\* password = "YOUR\_WIFI\_PASSWORD"; |
| ESP8266WebServer server(80); |
| String web\_on\_html = "<h1>IOTSERVER switch is ON</h1><p><a href=\"on\"><button>ON</button></a>&nbsp;<a  href=\"off\"><button>OFF</button></a></p>"; |
| String web\_off\_html = "<h1>IOTSERVER switch is OFF</h1><p><a href=\"on\"><button>ON</button></a>&nbsp;<a  href=\"off\"><button>OFF</button></a></p>"; |
| int gpio\_13\_led = 13; |
| int gpio\_12\_relay = 12; |
| void setup(void)  { |
| // First Time Initialisation |
| pinMode(gpio\_13\_led, OUTPUT); |
| digitalWrite(gpio\_13\_led, HIGH); |
| pinMode(gpio\_12\_relay, OUTPUT); |
| digitalWrite(gpio\_12\_relay, HIGH); |
| Serial.begin(115200); |
| delay(5000); |

|  |
| --- |
| WiFi.begin(ssid, password); |
| Serial.println("Connecting to wifi.."  ); |
| // Wait for connection |
| while (WiFi.status() != WL\_CONNECTED)  { |
| digitalWrite(gpio\_13\_led, LOW); |
| delay(500); |
| Serial.print("."); |
| Serial.println(WiFi.localIP()); |
| Serial.println(WiFi.status()); |
| digitalWrite(gpio\_13\_led, HIGH); |
| delay(500); |
| } |
| Serial.println(""); |
| Serial.print("Connected to "); |
| Serial.println(ssid); |
| Serial.print("IP address: "); |
| Serial.println(WiFi.localIP()); |
| if (mdns.begin("esp8266", WiFi.localIP()))  { |
| Serial.println("MDNS responder started"); |
| } |
| server.on("/", []()  { |
| if(digitalRead(gpio\_12\_relay)==HIGH)  { |
| server.send(200, "text/html", web\_on\_html); |

|  |
| --- |
| }  else  { |
| server.send(200, "text/html", web\_off\_html); |
| } |
| }; |
| server.on("/on", []()  { |
| server.send(200, "text/html", web\_on\_html); |
| digitalWrite(gpio\_13\_led, LOW); |
| digitalWrite(gpio\_12\_relay, HIGH); |
| delay(1000); |
| }; |
| server.on("/off", []()  { |
| server.send(200, "text/html", web\_off\_html); |
| digitalWrite(gpio\_13\_led, HIGH); |
| digitalWrite(gpio\_12\_relay, LOW); |
| delay(1000); |
| }; |
| server.begin(); |
| Serial.println("Server ready.."); |
| } |
| void loop(void)  { |
| server.handleClient(); |
| } |

**AUTHENTICATION CODING**

**// Performing API Authentication**

// Route change 0.0.0.0 mask 0.0.0.0 10.10.7.1

{

"deviceid":"Factory **xxxxx**",

"apikey":"**xxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx**", "accept":"post"

}

<http://10.10.7.1/ap>

{

"version": 2019,

"ssid": **[YOUR NETWORK SSID]**,

"password": **[YOUR NETWORK PASSWORD]**, "serverName": **[IP OF YOUR SERVER]**,

"port": **[PORT OF YOUR SERVER]**

}

{

"error": 0, "reason": "ok",

"IP": **[YOUR WEBSOCKET SERVER IP]**, "port": **[YOUR WEBSOCKET SERVER PORT]**

}

{

"error" : 0,

"deviceid" : **[ACTUAL DEVICE ID]**,

"apikey" : "**111111111-1111-1111-1111-111111111111**"

}

{

"error" : 0,

"date" : **[DATE IN ISO FORMAT]**, "deviceid" : **[ACTUAL DEVICE ID]**,

"apikey" : "**111111111-1111-1111-1111-111111111111**"

}

{action: 'update', value: {switch :state?'on':'off'}

{action: 'update', value: {timers :d.timers}

{

enabled : true,

type : **'once' OR 'repeat'**, at : time,

do :

{

switch : **'on' OR 'off'**

}}

const os = require('os'); const http = require('http'); const https = require('https'); const url = require('url'); const fs = require('fs');

var ws = require("nodejs-websocket"); var exec = require('child\_process').exec; var wlan = require('./wlan')();

var emitter = require('events').EventEmitter; var inherits = require('util').inherits; module.exports = IOTserver;

function IOTserver()

{

if (!(this instanceofIOTserver)) return new IOTserver(); emitter.call(this);

}

inherits(IOTserver, emitter); IOTserver.prototype.init = function init()

{

var self = this; self.\_initialized = false; self.\_connected = false; self.\_knownDevices = [];

self.\_inithttps(self);

};

IOTserver.prototype.pair = function (force, ssid, pwd)

{

var self = this;

var apSSID = "ITEAD-10000";

var find = setInterval(() => { if (!self.\_initialized) {

console.log('Waiting for initialization.'); return; //wait for init

}

wlan.Discover().then(nets =>

{

var apNet = nets.find(n =>n.ssid.startsWith(apSSID)); if (!apNet) {

console.log('ERR | IOTserver is not in pairing mode. Please, Long press until led start blinking fast.');

}

else

{

console.log('OK | IOTserver found in pairing mode.');

//apSSID = apNet.ssid; clearInterval(find);

if (self.\_nic.ssid != apNet.ssid) { wlan.Connect(apNet, '12345678').then(() =>

{

wlan.getNic().then(n =>

{

var nic = null;

if (n.length>= 1)

{

//get only first

nic = n[0];

else

{

} else {

console.log('ERR | No WLAN interfaces found. Unable to process.'); return;

}

if (nic.ssid != apNet.ssid) {

console.log('ERR | Unable to connect to the configuration AP.'); return;

}

\_initDevice(self, nic, self.\_nic.ssid, self.\_nic.key, force);

} };

}; }

else

{

console.log('ERR | You should not be connected to IOTserver configuration AP to pair device.');

} }};

}, 3000;

};

IOTserver.prototype.powerState = function (device, state)

{

var self = this;

return new Promise(function(resolve, reject)

{

var d = self.\_knownDevices.find(d=>d.id === device.id); if(!d)

{

reject('IOTserver device '+device.kind+' not found');

}

else

{

if(self.\_connected)

{

var h = f =>

{

if(f.device.id == d.id)

{

self.removeListener('msg',h); if(!f.err) resolve(f.device); else reject(f.err);

}

};

self.on('msg', h);

self.emit('push', {action: 'update', value: {switch : state?'on':'off'}, target: d.id});

} }};};

IOTserver.prototype.setTimer = function(device, time, state)

{

var self = this;

var d = self.\_knownDevices.find(d=>d.id === device.id); if(!d)

{

console.log('IOTserver device '+device.kind+' not found');}

else

{

if(self.\_connected)

{

d.timers = d.timers || []; d.timers.push(

{

enabled : true,

type : time.includes('T')?'once':'repeat', at : time,

do :

{

switch : state?'on':'off'

}

};

self.emit('push', {action: 'update', value: {timers : d.timers}, target: d.id};

} }

}

var \_initDevice = (self, nic, ssid, pwd, force) => {

exec("route change 0.0.0.0 mask 0.0.0.0 10.10.7.1", function (err, res) { if (err)

{

console.log('ERR | unable to set AP network: ', err); reject(err);

}

else

{

console.log('OK | '+res);

http.get('[http://10.10.7.1/device'](http://10.10.7.1/device%27), (res) =>

{

const sc = res.statusCode;

const ct = res.headers['content-type']; if (sc !== 200)

{

console.log('Unable to connect to the target device. Code: ' + sc);

res.resume(); return;

}

res.setEncoding('utf8'); var data = '';

res.on('data', (c) => data += c);

res.on('end', () =>

{

var response = JSON.parse(data); var device =

{

deviceid: response.deviceid, apikey : response.apikey

};

self.\_httpPost('[http://10.10.7.1/ap'](http://10.10.7.1/ap%27),

{

"version": 4,

"ssid": self.\_nic.ssid, "password": self.\_nic.key, "serverName": self.\_ip, "port": self.\_port

}, (re, err) =>

{

if (err)

{

console.log('Unable to configure endpoint ' + err);

}

else

{

console.log(JSON.stringify(re));

} };};

}.on('error', (e) =>

{

console.log(`Unable to establish connection to the device: ${e.message}`;

};

} };};

IOTserver.prototype.\_initServer = (self) =>

{

wlan.getNic().then(n =>

{

self.\_nic = n[0];

var ifaces = os.networkInterfaces(); for (var i in ifaces)

{

for (var k in ifaces[i])

{

var address = ifaces[i][k];

if (address.family === 'IPv4' && !address.internal&& address.mac ==

self.\_nic.mac)

{

self.\_ip = address.address; self.\_port = 80;

// Create Server

http.createServer((req, res) =>

{

res.writeHead(200,

{

'Content-Type': 'application/json'

};

res.end();

}.listen(self.\_port, self.\_ip, () =>

{

self.\_initialized = true;

};

break;

} }} };};

IOTserver.prototype.\_initws = (self, ip, port)=>

{

var options =

{secure : true,

key: fs.readFileSync('./tools/ipsum-key.pem'), cert: fs.readFileSync('./tools/ipsum-cert.pem'),

};

var server = ws.createServer(options,function (conn)

{

console.log("WS | Server is up %s:%s to

%s:%s",ip,port,conn.socket.remoteAddress,conn.socket.remotePort); self.\_connected = true;

self.on('push',a=>

{

var rq =

{

"apikey" : "111111111-1111-1111-1111-111111111111",

"action" : a.action, "deviceid" : a.target, "params" : a.value

};

var r = JSON.stringify(rq); console.log('REQ | WS | APP | ' + r);

conn.sendText(r);

};

conn.on("text", function (str)

{

var data = JSON.parse(str);

console.log('REQ | WS | DEV | %s', JSON.stringify(data)); res = {

"error" : 0,

"deviceid" : data.deviceid,

"apikey" : "111111111-1111-1111-1111-111111111111"

};

if(data.action)

{

switch(data.action)

{

case 'date':

res.date = new Date().toISOString(); break;

case 'query':

//device wants information

var device = self.\_knownDevices.find(d=>d.id == data.deviceid); if(!device)

{

console.log('ERR | WS | Unknown device ',data.deviceid);

}

else

{

/\*if(data.params.includes('timers'))

{

console.log('INFO | WS | Device %s asks for timers',device.id); if(device.timers){

res.params = [{timers : device.timers}];

}

}\*/ res.params = {};

data.params.forEach(p=>

{

res.params[p] = device[p];

};}

break;

case 'update':

//device wants to update its state

var device = self.\_knownDevices.find(d=>d.id == data.deviceid); if(!device)

{

console.log('ERR | WS | Unknown device ',data.deviceid);

}

else

{

device.state = data.params.switch; self.\_updateKnownDevice(self,device);

}

break;

case 'register': var device = {

id : data.deviceid

};

var type = data.deviceid.substr(0, 2); if(type == '01') device.kind = 'switch'; else if(type == '02') device.kind = 'light';

else if(type == '03') device.kind = 'sensor'; //temperature and humidity. No timers here;

device.version = data.romVersion; device.model = data.model;

self.\_updateKnownDevice(self,device);

console.log('INFO | WS | Device %s registered', device.id); break;

default: console.log('TODO | Unknown action "%s"',data.action);

break;

}

} else {

console.log('TODO | WS | Not data action frame');

}

var r = JSON.stringify(res); console.log('RES | WS | DEV | ' + r);

conn.sendText(r);

var td = self.\_knownDevices.find(d=>d.id == res.deviceid); self.emit('msg',{device : td});

};

conn.on("close", function (code, reason) { console.log("Connection closed");

};

}.listen(port,ip);

};

IOTserver.prototype.\_inithttps = (self)=>

{

wlan.getNic().then(n =>

{

self.\_nic = n[0];

var ifaces = os.networkInterfaces(); for (var i in ifaces) {

for (var k in ifaces[i])

{

var address = ifaces[i][k];

if (address.family === 'IPv4' && !address.internal&& address.mac ==

self.\_nic.mac)

{

self.\_ip = address.address; self.\_port = 80;

self.\_initws(self,self.\_ip,self.\_port + 1);

const options = {

key: fs.readFileSync('./tools/ipsum-key.pem'), cert: fs.readFileSync('./tools/ipsum-cert.pem'),

};

var server = https.createServer(options, (req, res) =>

{

console.log('REQ | %s | %s ',req.method, req.url); var body = [];

req.on('data', function(chunk)

{

body.push(chunk);

}.on('end', function()

{

body = JSON.parse(Buffer.concat(body).toString('utf-8')); console.log('REQ | %s',JSON.stringify(body));

res.writeHead(200); res.end(JSON.stringify(

{

"error": 0, "reason": "ok",

"IP": self.\_ip, "port": self.\_port + 1

};

};

}.listen(self.\_port,self.\_ip); server.on('connection', c=>

{

console.log("Connection: %s:%s",c.remoteAddress, c.remotePort);

callback(response);

});

}).on('error', (e) =>

{

console.log(`unable to post request: ${e.message}`); callback(null, e);

};

req.write(dta); req.end();

};

IOTserver.prototype.\_updateKnownDevice = (self, device) =>

{

var updated = false;

for (var i = 0; i<self.\_knownDevices.length; i++)

{

if (self.\_knownDevices[i].id == device.id)

{

self.\_knownDevices[i] = device; updated = true;

self.emit('deviceUpdated',device);

}

}

if (!updated)

{

self.\_knownDevices.push(device); self.emit('deviceAdded',device);

}

};

|  |
| --- |
| Message | Unit | Description |
| | | |
| TotalStartTime | Date | DateTime of calculation for Total Power  Total | kWh | Total Energy usage including Today  Yesterday | kWh | Total Energy usage between 00:00 and 24:00 yesterday  Today | kWh | Total Energy usage today from 00:00 until now  Period | Wh | Energy usage between previous message and now  Power | W | Current effective power load  ApparentPower | W | Power load on the cable = sqrt(Power^2 + ReactivePower^2)  Factor | | The ratio of the real power flowing to the load to  Voltage | V | Current line voltage  Current | A | Current line current |

**IOT dashboard coding**

// IOT Dashboard Settings #ifndef \_SETTINGS\_H\_

|  |
| --- |
| #definePARAM8\_SIZE18// Number of param bytes (SetOption) |
| typedefunion{ // Restricted by MISRA-C Rule 18.4 but so useful... |
| uint32\_tdata; // Allow bit manipulation using SetOption |
| struct{ // SetOption0 .. SetOption31 |
| uint32\_tsave\_state :1; // bit 0 - SetOption0 - Save power state and  use after restart |
| uint32\_tbutton\_restrict :1; // bit 1 - SetOption1 - Control button  multipress |
| uint32\_tvalue\_units :1; // bit 2 - SetOption2 - Add units to JSON  status messages |
| uint32\_tmqtt\_enabled :1; // bit 3 - SetOption3 - Control MQTT |
| uint32\_tmqtt\_response :1; // bit 4 - SetOption4 - Switch between  MQTT RESULT or COMMAND |
| uint32\_tmqtt\_power\_retain :1; // bit 5 - CMND\_POWERRETAIN |
| uint32\_tmqtt\_button\_retain :1; // bit 6 - CMND\_BUTTONRETAIN |
| uint32\_tmqtt\_switch\_retain :1; // bit 7 - CMND\_SWITCHRETAIN |
| uint32\_ttemperature\_conversion :1; // bit 8 - SetOption8 - Switch  between Celsius or Fahrenheit |
| uint32\_tmqtt\_sensor\_retain :1; // bit 9 - CMND\_SENSORRETAIN |
| uint32\_tmqtt\_offline :1; // bit 10 - SetOption10 - Control MQTT  LWT message format |
| uint32\_tbutton\_swap :1; // bit 11 (v5.1.6) - SetOption11 - Swap button  single and double press functionality |
| uint32\_tstop\_flash\_rotate :1; // bit 12 (v5.2.0) - SetOption12 - Switch  between dynamic or fixed slot flash save location |
| uint32\_tbutton\_single :1; // bit 13 (v5.4.0) - SetOption13 - Support only  single press to speed up button press recognition |
| uint32\_tinterlock :1; // bit 14 (v5.6.0) - CMND\_INTERLOCK |

|  |
| --- |
| uint32\_tpwm\_control :1; // bit 15 (v5.8.1) - SetOption15 - Switch  between commands PWM or COLOR/DIMMER/CT/CHANNEL |
| uint32\_tws\_clock\_reverse :1; // bit 16 (v5.8.1) - SetOption16 - Switch  between clockwise or counter-clockwise |
| uint32\_tdecimal\_text :1; // bit 17 (v5.8.1) - SetOption17 - Switch between  decimal or hexadecimal output |
| uint32\_tlight\_signal :1; // bit 18 (v5.10.0c) - SetOption18 - Pair light signal  with CO2 sensor |
| uint32\_thass\_discovery :1; // bit 19 (v5.11.1a) - SetOption19 - Control Home  Assistantautomatic discovery (See SetOption59) |
| uint32\_tnot\_power\_linked :1; // bit 20 (v5.11.1f) - SetOption20 - Control  power in relation to Dimmer/Color/Ct changes |
| uint32\_tno\_power\_on\_check :1; // bit 21 (v5.11.1i) - SetOption21 - Show  voltage even if powered off |
| uint32\_tmqtt\_serial :1; // bit 22 (v5.12.0f) - CMND\_SERIALSEND and  CMND\_SERIALLOG |
| uint32\_tmqtt\_serial\_raw :1; // bit 23 (v6.1.1c) - CMND\_SERIALSEND3 |
| uint32\_tpressure\_conversion :1; // bit 24 (v6.3.0.2) - SetOption24 - Switch  between hPa or mmHg pressure unit |
| uint32\_tknx\_enabled :1; // bit 25 (v5.12.0l) - CMND\_KNX\_ENABLED |
| uint32\_tdevice\_index\_enable :1; // bit 26 (v5.13.1a) - SetOption26 - Switch  between POWER or POWER1 |
| uint32\_tknx\_enable\_enhancement :1; // bit 27 (v5.14.0a) -  CMND\_KNX\_ENHANCED |
| uint32\_trf\_receive\_decimal :1; // bit 28 (v6.0.0a) - SetOption28 - RF receive  data format |
| uint32\_tir\_receive\_decimal :1; // bit 29 (v6.0.0a) - SetOption29 - IR receive  data format |

|  |
| --- |
| uint32\_thass\_light :1; // bit 30 (v6.0.0b) - SetOption30 - Enforce  HAssautodiscovery as light |
| uint32\_tglobal\_state :1; // bit 31 (v6.1.0) - SetOption31 - Control link led  blinking |
| }; |
| } SysBitfield; |
| typedefunion{ // Restricted by MISRA-C Rule 18.4 but so useful... |
| uint32\_tdata; // Allow bit manipulation using SetOption |
| struct{ // SetOption50 .. SetOption81 |
| uint32\_ttimers\_enable :1; // bit 0 (v6.1.1b) - CMND\_TIMERS |
| uint32\_t user\_esp8285\_enable :1; // bit 1 (v6.1.1.14) - SetOption51 - Enable  ESP8285 user GPIO's |
| uint32\_ttime\_append\_timezone :1; // bit 2 (v6.2.1.2) - SetOption52 - Append  timezone to JSON time |
| uint32\_tgui\_hostname\_ip :1; // bit 3 (v6.2.1.20) - SetOption53 - Show  hostanme and IP address in GUI main menu |
| uint32\_t tuya\_apply\_o20 :1; // bit 4 (v6.3.0.4) - SetOption54 - Apply  SetOption20 settings to Tuya device |
| uint32\_tmdns\_enabled :1; // bit 5 (v6.4.1.4) - SetOption55 - Control  mDNS service |
| uint32\_tuse\_wifi\_scan :1; // bit 6 (v6.3.0.10) - SetOption56 - Scan wifi  network at restart for configured AP's |
| uint32\_tuse\_wifi\_rescan :1; // bit 7 (v6.3.0.10) - SetOption57 - Scan wifi  network every 44 minutes for configured AP's |
| uint32\_treceive\_raw :1; // bit 8 (v6.3.0.11) - SetOption58 - Add IR Raw  data to JSON message |
| uint32\_thass\_tele\_on\_power :1; // bit 9 (v6.3.0.13) - SetOption59 - Send  tele/%topic%/STATE in addition to stat/%topic%/RESULT |

|  |
| --- |
| uint32\_tsleep\_normal :1; // bit 10 (v6.3.0.15) - SetOption60 - Enable normal  sleep instead of dynamic sleep |
| uint32\_tbutton\_switch\_force\_local :1;// bit 11 (v6.3.0.16) - SetOption61 - Force  local operation when button/switch topic is set |
| uint32\_tno\_hold\_retain :1; // bit 12 (v6.4.1.19) - SetOption62 - Don't use  retain flag on HOLD messages |
| uint32\_t spare13 :1; |
| uint32\_t spare14 :1; |
| uint32\_t spare15 :1; |
| uint32\_t spare16 :1; |
| uint32\_t spare17 :1; |
| uint32\_t spare18 :1; |
| uint32\_t spare19 :1; |
| uint32\_t spare20 :1; |
| uint32\_t spare21 :1; |
| uint32\_t spare22 :1; |
| uint32\_t spare23 :1; |
| uint32\_t spare24 :1; |
| uint32\_t spare25 :1; |
| uint32\_t spare26 :1; |
| uint32\_t spare27 :1; |
| uint32\_t spare28 :1; |
| uint32\_t spare29 :1; |
| uint32\_t spare30 :1; |
| uint32\_t spare31 :1; |
| }; |
| } SysBitfield3; |
| typedefunion { |

|  |
| --- |
| uint32\_tdata; // Allow bit manipulation |
| struct { |
| uint32\_t spare00 :1; |
| uint32\_t spare01 :1; |
| uint32\_t spare02 :1; |
| uint32\_t spare03 :1; |
| uint32\_t spare04 :1; |
| uint32\_t spare05 :1; |
| uint32\_tcalc\_resolution :3; |
| uint32\_tweight\_resolution :2; |
| uint32\_tfrequency\_resolution :2; |
| uint32\_taxis\_resolution :2; |
| uint32\_tcurrent\_resolution :2; |
| uint32\_tvoltage\_resolution :2; |
| uint32\_twattage\_resolution :2; |
| uint32\_temulation :2; |
| uint32\_tenergy\_resolution :3; |
| uint32\_tpressure\_resolution :2; |
| uint32\_thumidity\_resolution :2; |
| uint32\_ttemperature\_resolution :2; |
| }; |
| } SysBitfield2; |
| typedefunion { |
| uint16\_t data; |
| struct { |
| uint16\_themis :1; // bit 0 = 0=Northern, 1=Southern Hemisphere  (=Opposite DST/STD) |

|  |
| --- |
| uint16\_tweek :3; // bits 1 - 3 = 0=Last week of the month, 1=First,  2=Second, 3=Third, 4=Fourth |
| uint16\_tmonth :4; // bits 4 - 7 = 1=Jan, 2=Feb, ... 12=Dec |
| uint16\_tdow :3; // bits 8 - 10 = day of week, 1=Sun, 2=Mon, ... 7=Sat |
| uint16\_thour :5; // bits 11 - 15 = 0-23 |
| }; |
| } TimeRule; |
| typedefunion { |
| uint32\_t data; |
| struct { |
| uint32\_ttime :11; // bits 0 - 10 = minutes in a day |
| uint32\_twindow :4; // bits 11 - 14 = minutes random window |
| uint32\_trepeat :1; // bit 15 |
| uint32\_tdays :7; // bits 16 - 22 = week day mask |
| uint32\_tdevice :4; // bits 23 - 26 = 16 devices |
| uint32\_tpower :2; // bits 27 - 28 = 4 power states - Off, On, Toggle,  Blink or Rule |
| uint32\_tmode :2; // bits 29 - 30 = timer modes - 0 = Scheduler, 1 =  Sunrise, 2 = Sunset |
| uint32\_tarm :1; // bit 31 |
| }; |
| } Timer; |
| typedefunion { |
| uint16\_t data; |
| struct { |
| uint16\_tpinmode :3; // Pin mode (1 through 6) |
| uint16\_tpullup :1; // Enable internal weak pull-up resistor |
| uint16\_tsaved\_state :1; // Save output state, if used. |

|  |
| --- |
| uint16\_tint\_report\_mode :2; // Interrupt reporting mode 0 = immediate  telemetry & event, 1 = immediate event only, 2 = immediate telemetry only |
| uint16\_tint\_report\_defer :4; // Number of interrupts to ignore until reporting  (default 0, max 15) |
| uint16\_tint\_count\_en :1; // Enable interrupt counter for this pin |
| uint16\_tint\_retain\_flag :1; // Report if interrupt occured for pin in next  teleperiod |
| uint16\_t spare13 :1; |
| uint16\_t spare14 :1; |
| uint16\_t spare15 :1; |
| }; |
| } Mcp230xxCfg; |
| typedefunion { |
| uint8\_t data; |
| struct { |
| uint8\_t spare0 :1; |
| uint8\_t spare1 :1; |
| uint8\_t spare2 :1; |
| uint8\_t spare3 :1; |
| uint8\_t spare4 :1; |
| uint8\_t spare5 :1; |
| uint8\_t spare6 :1; |
| uint8\_t mhz19b\_abc\_disable :1; // Disable ABC (Automatic Baseline Correction for MHZ19(B) (0 = Enabled (default), 1 = Disabled with Sensor15  command) |
| }; |
| } SensorCfg1; |
| /\* |

|  |
| --- |
| struct SYSCFG { |
| unsigned long cfg\_holder; // 000 Pre v6 header |
| unsigned long save\_flag; // 004 |
| unsigned long version; // 008 |
| unsigned long bootcount; // 00C |
| \*/ |
| struct SYSCFG { |
| uint16\_tcfg\_holder; // 000 v6 header |
| uint16\_tcfg\_size; // 002 |
| unsignedlongsave\_flag; // 004 |
| unsignedlongversion; // 008 |
| uint16\_tbootcount; // 00C |
| uint16\_tcfg\_crc; // 00E |
| SysBitfieldflag; // 010 |
| int16\_tsave\_data; // 014 |
| int8\_ttimezone; // 016 |
| charota\_url[101]; // 017 |
| charmqtt\_prefix[3][11]; // 07C |
| uint8\_tbaudrate; // 09D |
| uint8\_tseriallog\_level; // 09E |
| uint8\_tsta\_config; // 09F |
| uint8\_tsta\_active; // 0A0 |
| charsta\_ssid[2][33]; // 0A1 - Keep together with sta\_pwd as being copied as  one chunck with reset 4/5 |
| charsta\_pwd[2][65]; // 0E3 - Keep together with sta\_ssid as being copied as  one chunck with reset 4/5 |
| charhostname[33]; // 165 |
| charsyslog\_host[33]; // 186 |

|  |
| --- |
| uint8\_trule\_stop; // 1A7 |
| uint16\_tsyslog\_port; // 1A8 |
| uint8\_tsyslog\_level; // 1AA |
| uint8\_twebserver; // 1AB |
| uint8\_tweblog\_level; // 1AC |
| uint8\_tmqtt\_fingerprint[2][20]; // 1AD |
| uint8\_t free\_1D5[20]; // 1D5 Free since 5.12.0e |
| charmqtt\_host[33]; // 1E9 |
| uint16\_tmqtt\_port; // 20A |
| charmqtt\_client[33]; // 20C |
| charmqtt\_user[33]; // 22D |
| charmqtt\_pwd[33]; // 24E |
| charmqtt\_topic[33]; // 26F |
| charbutton\_topic[33]; // 290 |
| charmqtt\_grptopic[33]; // 2B1 |
| uint8\_tdisplay\_model; // 2D2 |
| uint8\_tdisplay\_mode; // 2D3 |
| uint8\_tdisplay\_refresh; // 2D4 |
| uint8\_tdisplay\_rows; // 2D5 |
| uint8\_tdisplay\_cols[2]; // 2D6 |
| uint8\_tdisplay\_address[8]; // 2D8 |
| uint8\_tdisplay\_dimmer; // 2E0 |
| uint8\_tdisplay\_size; // 2E1 |
| TimeRuletflag[2]; // 2E2 |
| uint16\_tpwm\_frequency; // 2E6 |
| power\_tpower; // 2E8 |
| uint16\_tpwm\_value[MAX\_PWMS]; // 2EC |
| int16\_taltitude; // 2F6 |

|  |
| --- |
| uint16\_ttele\_period; // 2F8 |
| uint8\_tdisplay\_rotate; // 2FA |
| uint8\_tledstate; // 2FB |
| uint8\_tparam[PARAM8\_SIZE]; // 2FC SetOption32 .. SetOption49 |
| int16\_ttoffset[2]; // 30E |
| uint8\_tdisplay\_font; // 312 |
| charstate\_text[4][11]; // 313 |
| uint8\_tenergy\_power\_delta; // 33F |
| uint16\_tdomoticz\_update\_timer; // 340 |
| uint16\_tpwm\_range; // 342 |
| unsignedlongdomoticz\_relay\_idx[MAX\_DOMOTICZ\_IDX]; // 344 |
| unsignedlongdomoticz\_key\_idx[MAX\_DOMOTICZ\_IDX]; // 354 |
| unsignedlongenergy\_power\_calibration; // 364 |
| unsignedlongenergy\_voltage\_calibration; // 368 |
| unsignedlongenergy\_current\_calibration; // 36C |
| unsignedlongenergy\_kWhtoday; // 370 |
| unsignedlongenergy\_kWhyesterday; // 374 |
| uint16\_tenergy\_kWhdoy; // 378 |
| uint16\_tenergy\_min\_power; // 37A |
| uint16\_tenergy\_max\_power; // 37C |
| uint16\_tenergy\_min\_voltage; // 37E |
| uint16\_tenergy\_max\_voltage; // 380 |
| uint16\_tenergy\_min\_current; // 382 |
| uint16\_tenergy\_max\_current; // 384 |
| uint16\_tenergy\_max\_power\_limit; // 386 MaxPowerLimit |
| uint16\_tenergy\_max\_power\_limit\_hold; // 388 MaxPowerLimitHold |
| uint16\_tenergy\_max\_power\_limit\_window; // 38A MaxPowerLimitWindow |
| uint16\_tenergy\_max\_power\_safe\_limit; // 38C MaxSafePowerLimit |

|  |
| --- |
| uint16\_tenergy\_max\_power\_safe\_limit\_hold; // 38E MaxSafePowerLimitHold |
| uint16\_tenergy\_max\_power\_safe\_limit\_window; // 390  MaxSafePowerLimitWindow |
| uint16\_tenergy\_max\_energy; // 392 MaxEnergy |
| uint16\_tenergy\_max\_energy\_start; // 394 MaxEnergyStart |
| uint16\_tmqtt\_retry; // 396 |
| uint8\_tpoweronstate; // 398 |
| uint8\_tlast\_module; // 399 |
| uint16\_tblinktime; // 39A |
| uint16\_tblinkcount; // 39C |
| uint16\_tlight\_rotation; // 39E |
| SysBitfield3 flag3; // 3A0 |
| uint8\_tswitchmode[MAX\_SWITCHES]; // 3A4 (6.0.0b - moved from 0x4CA) |
| charfriendlyname[MAX\_FRIENDLYNAMES][33]; // 3AC |
| charswitch\_topic[33]; // 430 |
| charserial\_delimiter; // 451 |
| uint8\_tsbaudrate; // 452 |
| uint8\_tsleep; // 453 |
| uint16\_tdomoticz\_switch\_idx[MAX\_DOMOTICZ\_IDX]; // 454 |
| uint16\_tdomoticz\_sensor\_idx[MAX\_DOMOTICZ\_SNS\_IDX]; // 45C |
| uint8\_tmodule; // 474 |
| uint8\_tws\_color[4][3]; // 475 |
| uint8\_tws\_width[3]; // 481 |
| myiomy\_gp; // 484 |
| uint8\_ttest\_step; // 495 |
| uint16\_tlight\_pixels; // 496 |
| uint8\_tlight\_color[5]; // 498 |
| uint8\_tlight\_correction; // 49D |

|  |
| --- |
| uint8\_tlight\_dimmer; // 49E |
| uint8\_trule\_enabled; // 49F |
| uint8\_trule\_once; // 4A0 |
| uint8\_tlight\_fade; // 4A1 |
| uint8\_tlight\_speed; // 4A2 |
| uint8\_tlight\_scheme; // 4A3 |
| uint8\_tlight\_width; // 4A4 |
| uint8\_tknx\_GA\_registered; // 4A5 Number of Group Address to read |
| uint16\_tlight\_wakeup; // 4A6 |
| uint8\_tknx\_CB\_registered; // 4A8 Number of Group Address to write |
| charweb\_password[33]; // 4A9 |
| uint8\_tinterlock[MAX\_INTERLOCKS]; // 4CA |
| charntp\_server[3][33]; // 4CE |
| uint8\_t ina219\_mode; // 531 |
| uint16\_tpulse\_timer[MAX\_PULSETIMERS]; // 532 |
| uint16\_tbutton\_debounce; // 542 |
| uint32\_tip\_address[4]; // 544 |
| unsignedlongenergy\_kWhtotal; // 554 |
| charmqtt\_fulltopic[100]; // 558 |
| SysBitfield2 flag2; // 5BC |
| unsignedlongpulse\_counter[MAX\_COUNTERS]; // 5C0 |
| uint16\_tpulse\_counter\_type; // 5D0 |
| uint16\_tpulse\_counter\_debounce; // 5D2 |
| uint8\_trf\_code[17][9]; // 5D4 |
| uint8\_ttimezone\_minutes; // 66D |
| uint16\_tswitch\_debounce; // 66E |
| Timer timer[MAX\_TIMERS]; // 670 |
| intlatitude; // 6B0 |

|  |
| --- |
| intlongitude; // 6B4 |
| uint16\_tknx\_physsical\_addr; // 6B8 (address\_t is a uint16\_t) |
| uint16\_tknx\_GA\_addr[MAX\_KNX\_GA]; // 6BA (address\_t is a uint16\_t) x  KNX\_max\_GA |
| uint16\_tknx\_CB\_addr[MAX\_KNX\_CB]; // 6CE (address\_t is a uint16\_t) x  KNX\_max\_CB |
| uint8\_tknx\_GA\_param[MAX\_KNX\_GA]; // 6E2 Type of Input (relay changed,  button pressed, sensor read <-teleperiod) |
| uint8\_tknx\_CB\_param[MAX\_KNX\_CB]; // 6EC Type of Output (set relay, toggle  relay, reply sensor value) |
| Mcp230xxCfg mcp230xx\_config[16]; // 6F6 |
| uint8\_t mcp230xx\_int\_prio; // 716 |
| SensorCfg1 SensorBits1; // 717 On/Off settings used by Sensor Commands |
| uint16\_t mcp230xx\_int\_timer; // 718 |
| uint8\_trgbwwTable[5]; // 71A |
| uint8\_tuser\_template\_base; // 71F |
| mytmpltuser\_template; // 720 29 bytes |
| uint8\_t free\_73D[87]; // 73D |
| uint32\_tdrivers[3]; // 794 |
| uint32\_tmonitors; // 7A0 |
| uint32\_tsensors[3]; // 7A4 |
| uint32\_tdisplays; // 7B0 |
| uint32\_tenergy\_kWhtotal\_time; // 7B4 |
| unsignedlongweight\_item; // 7B8 Weight of one item in gram \* 10 |
| uint8\_t free\_7BC[2]; // 7BC |
| uint16\_tweight\_max; // 7BE Total max weight in kilogram |
| unsignedlongweight\_reference; // 7C0 Reference weight in gram |
| unsignedlongweight\_calibration; // 7C4 |

|  |
| --- |
| unsignedlongenergy\_frequency\_calibration; // 7C8 |
| uint16\_tweb\_refresh; // 7CC |
| charmems[MAX\_RULE\_MEMS][10]; // 7CE |
| charrules[MAX\_RULE\_SETS][MAX\_RULE\_SIZE]; // 800 uses 512 bytes in  v5.12.0m, 3 x 512 bytes in v5.14.0b |
| // E00 - FFF free locations |
| } Settings; |
| struct RTCRBT { |
| uint16\_tvalid; // 280 (RTC memory offset 100 - sizeof(RTCRBT)) |
| uint8\_tfast\_reboot\_count; // 282 |
| uint8\_t free\_003[1]; // 283 |
| } RtcReboot; |
| struct RTCMEM { |
| uint16\_tvalid; // 290 (RTC memory offset 100) |
| uint8\_toswatch\_blocked\_loop; // 292 |
| uint8\_tota\_loader; // 293 |
| unsignedlongenergy\_kWhtoday; // 294 |
| unsignedlongenergy\_kWhtotal; // 298 |
| unsignedlongpulse\_counter[MAX\_COUNTERS]; // 29C |
| power\_tpower; // 2AC |
| uint8\_t free\_020[60]; // 2B0 |
| // 2EC - 2FF free locations |
| } RtcSettings; |
| struct TIME\_T { |
| uint8\_t second; |
| uint8\_t minute; |
| uint8\_t hour; |
| uint8\_tday\_of\_week; // sunday is day 1 |

|  |
| --- |
| uint8\_tday\_of\_month; |
| uint8\_t month; |
| charname\_of\_month[4]; |
| uint16\_tday\_of\_year; |
| uint16\_t year; |
| unsignedlong days; |
| unsignedlong valid; |
| } RtcTime; |
| struct XDRVMAILBOX { |
| uint16\_t valid; |
| uint16\_tindex; |
| uint16\_tdata\_len; |
| uint16\_t payload16; |
| int16\_t payload; |
| boolgrpflg; |
| uint8\_tnotused; |
| char \*topic; |
| char \*data; |
| } XdrvMailbox; |
| #defineMAX\_RULES\_FLAG7// Number of bits used in RulesBitfield (tricky I  know...) |
| typedefunion{ // Restricted by MISRA-C Rule 18.4 but so useful... |
| uint16\_tdata; // Allow bit manipulation |
| struct { |
| uint16\_tsystem\_boot :1; |
| uint16\_ttime\_init :1; |
| uint16\_ttime\_set :1; |
| uint16\_tmqtt\_connected :1; |

|  |
| --- |
| uint16\_tmqtt\_disconnected :1; |
| uint16\_twifi\_connected :1; |
| uint16\_twifi\_disconnected :1; |
| uint16\_t spare07 :1; |
| uint16\_t spare08 :1; |
| uint16\_t spare09 :1; |
| uint16\_t spare10 :1; |
| uint16\_t spare11 :1; |
| uint16\_t spare12 :1; |
| uint16\_t spare13 :1; |
| uint16\_t spare14 :1; |
| uint16\_t spare15 :1; |
| }; |
| } RulesBitfield; |
| typedefunion { |
| uint8\_t data; |
| struct { |
| uint8\_twifi\_down :1; |
| uint8\_tmqtt\_down :1; |
| uint8\_t spare02 :1; |
| uint8\_t spare03 :1; |
| uint8\_t spare04 :1; |
| uint8\_t spare05 :1; |
| uint8\_t spare06 :1; |
| uint8\_t spare07 :1; |
| }; |
| } StateBitfield; |

|  |
| --- |
| **// Over Load Protection Service** |
| Settings.flag2.current\_resolution = 3; |
| // Settings.flag2.voltage\_resolution = 0; |
| // Settings.flag2.wattage\_resolution = 0; |
| Settings.flag2.energy\_resolution = ENERGY\_RESOLUTION; |
| Settings.param[P\_MAX\_POWER\_RETRY] = MAX\_POWER\_RETRY; |
| Settings.energy\_power\_delta = DEFAULT\_POWER\_DELTA; |
| Settings.energy\_power\_calibration = HLW\_PREF\_PULSE; |
| Settings.energy\_voltage\_calibration = HLW\_UREF\_PULSE; |
| Settings.energy\_current\_calibration = HLW\_IREF\_PULSE; |
| // Settings.energy\_kWhtoday = 0; |
| // Settings.energy\_kWhyesterday = 0; |
| // Settings.energy\_kWhtdy = 0; |
| // Settings.energy\_min\_power = 0; |
| // Settings.energy\_max\_power = 0; |
| // Settings.energy\_min\_voltage = 0; |
| // Settings.energy\_max\_voltage = 0; |
| // Settings.energy\_min\_current = 0; |
| // Settings.energy\_max\_current = 0; |
| // Settings.energy\_max\_power\_limit = 0; // MaxPowerLimit |
| Settings.energy\_max\_power\_limit\_hold = MAX\_POWER\_HOLD; |
| Settings.energy\_max\_power\_limit\_window = MAX\_POWER\_WINDOW; |
| // Settings.energy\_max\_power\_safe\_limit = 0; //  MaxSafePowerLimit |
| Settings.energy\_max\_power\_safe\_limit\_hold = SAFE\_POWER\_HOLD; |
| Settings.energy\_max\_power\_safe\_limit\_window =  SAFE\_POWER\_WINDOW; |
| // Settings.energy\_max\_energy = 0; // MaxEnergy |

|  |
| --- |
| // Settings.energy\_max\_energy\_start = 0; // MaxEnergyStart |
| // Settings.energy\_kWhtotal = 0; |
| RtcSettings.energy\_kWhtotal = 0; |