Typografische Konventionen für C

Normal Text	Control Flow	Keyword	Data Type	Decimal	Octal
Hex	Binary	Float	Standard Suffix	Char	String
String Char	Comment	Symbol	Preprocessor	Prep. Lib	Region Marker
Error					

Doxygen Text

Normal Text	Tags	Custom Tags	Word	HTML Tag	*Entities*
Description	Comment	Region	Identifier	HTML Comment	Types
Code	*Dot Graph*	*Formulas*	*Message Sequence Chart*	Verbatim	Note
Warning	Attention	Todo			

Alerts Text

Normal Text	Alert Level 1	Alert Level 2	Alert Level 3	Region Marker

```
/* INDIDUINOMETEO FIRMWARE.
  NACHO MAS 2013. http://indiduino.wordpress.com
  Magnus W. Eriksen 2017 https://github.com/magnue

  Several modifications over indiduinoTemplate:
  .- Include  "i2cmaster.h",  "Adafruit_BMP085.h", "Adafruit_MLX90614.h" and  "dht.h" libraries
   to read the sensors.
  .- Add custimization of pinnumbers, and flags for frezzing and daylight.
  .- Allow user to disable any sensor(s) and its (their) code when compiling.
  .- Several additional functions to read the sensor and calculate flags,
   cloudcover and dew point.
  .- Extend event loop fail checks to awoid reading values from unreadable sensors (IR, P and DHT).
  .- Overwrite firmata TOTAL_ANALOG_PINS and TOTAL_PINS to make room for
   aditional (>6) analog calculate inputs.
  .- Use pullup resitor on inputs for signal flags.
  .- Made new firmware version to make use of recent sensors (BME280 and TSL2591). G. Gagnon 2019
  .- Added BME280 support (BME280 includes Barometric pressure, humidity and temperature sensors)
     -> Removed BMP and DHT G.Gagnon 2019
  .- Added TSL2591 support to replace irradiance sensor. Provides estimate of Sky Quality. G. Gagnon 2019
  .- BME280 and TSL2591 are I2C sensors that connect the same as the MLX90614, through an I2C bus. It is recommended to get
    3.3 volts parts along with the Arduino (Arduino Pro Mini is available in a 3.3V version) to avoid having to use level
    shifters.

  IMPORTANT: Customize following values to match your setup
*/

// Comment out if your setup don't include some sensor.Some sensor combinations are exclusive; BME280 replaces both
// DHT and BMP series sensors, TSL replaces the irradiance sensor but could be used alongside it is the firmware is
// modified appropriately.
//#define USE_DHT_SENSOR          //USE DHT HUMITITY SENSOR. Comment if not.
#define USE_MLX_SENSOR            //USE MELEXIS MLX90614IR SENSOR. Comment if not.
#define USE_BME_SENSOR            //USE BME280 ENVIRONMENT SENSOR. Comment if not.
#define USE_TSL_SENSOR            //USE TSL2591 SENSOR. Comment if not.
//define USE_P_SENSOR            //USE BMP085 PRESSURE SENSOR. Comment if not.
```

```
35 //#define USE_IRRADIANCE_SENSOR   //USE IRRADIANCE SENSOR (solar cell). Comment if not.
36
37 #if defined USE_P_SENSOR && defined USE_BME_SENSOR
38   #error Choose either USE_BME_SENSOR or USE_P_SENSOR
39 #endif
40
41 #if defined USE_DHT_SENSOR && defined USE_BME_SENSOR
42   #error Choose either USE_BME_SENSOR or USE_DHT_SENSOR
43 #endif
44
45 #if defined USE_TSL_SENSOR && defined USE_IRRADIANCE_SENSOR
46   #error Choose either USE_TSL_SENSOR or USE_IRRADIANCE_SENSOR
47 #endif
48
49 //Not everyone consider zero celsius as frezzing and other drivers will react to frezzing as an alert.
50 //define temperature limit for issuing alert.
51 //Default 0
52 #define FREZZING 0
53
54 //All sensors (Thr=DHT22,Tir=MELEXIS and Tp=BME280) include a ambient temperature
55 //Chosse  that sensor, only one, is going to use for main Ambient Temperature:
56 //#define T_MAIN_Thr
57 //#define T_MAIN_Tir
58 #define T_MAIN_Tp
59
60 #ifdef USE_IRRADIANCE_SENSOR
61   //A multitude of solar cells can be used as IRRADIANCE sensor.
62   //Set MINIMUM_DAYLIGHT to the IRRADIANCE output at start of dusk.
63   #define MINIMUM_DAYLIGHT 100
64
65    // what analog pin we connected IRRADCIANCE to
66   #define IR_RADIANCE_PIN 0
67 #endif //USE_IRRADIANCE_SENSOR
68
69 #ifdef USE_TSL_SENSOR
70 //Set DAYLIGHT to the SQM output at start of dusk. 15.0 is probably a reasonnable value.
71 #define DAYLIGHT 15.0
72 #endif //USE_TSL_SENSOR
73
74 #ifdef USE_MLX_SENSOR
75 //Cloudy sky is warmer that clear sky. Thus sky temperature meassure by IR sensor
76 //is a good indicator to estimate cloud cover. However IR really meassure the
77 //temperatura of all the air column above increassing with ambient temperature.
78 //So it is important include some correction factor:
79 //From AAG Cloudwatcher formula. Need to improve futher.
80 //http://www.aagware.eu/aag/cloudwatcher700/WebHelp/index.htm#page=Operational%20Aspects/23-TemperatureFactor-.htm
81 //Sky temp correction factor. Tsky=Tsky_meassure - Tcorrection
82 //Formula Tcorrection = (K1 / 100) * (Thr - K2 / 10) + (K3 / 100) * pow((exp (K4 / 1000* Thr)) , (K5 / 100));
```

```
#define  K1 33.
#define  K2 0.
#define  K3 4.
#define  K4 100.
#define  K5 100.

//Clear sky corrected temperature (temp below means 0% clouds)
#define CLOUD_TEMP_CLEAR  -8
//Totally cover sky corrected temperature (temp above means 100% clouds)
#define CLOUD_TEMP_OVERCAST  0
//Activation treshold for cloudFlag (%)
#define CLOUD_FLAG_PERCENT  30
#endif //USE_MLX_SENSOR

#define IPS_OK    1
#define IPS_ALERT 3

// Having only two states covering IPS_IDLE, IPS_OK, ... is problematic
// We map IPS_OK to 1 and everything else to 0

#define STATUS_OK HIGH
#define STATUS_NOT_OK LOW

/*END OFF CUSTOMITATION. YOU SHOULT NOT NEED TO CHANGE ANYTHING BELOW */

/*
 * Pin settings
*/
#define PIN_STATUS_CLOUDY   2
#define PIN_STATUS_DEW      4
#define PIN_STATUS_FREZZY   5
#define PIN_STATUS_DAYLIGHT 6
#define PIN_STATUS_SQM      6
#define PIN_STATUS_MLX      7
#define PIN_STATUS_TSL      8
#define PIN_STATUS_BMP      9
#define PIN_STATUS_BME      9

/*
   Firmata is a generic protocol for communicating with microcontrollers
   from software on a host computer. It is intended to work with
   any host computer software package.

   To download a host software package, please clink on the following link
   to open the download page in your default browser.

   http://firmata.org/wiki/Download
*/
```

```
131
132 /*
133   Copyright (C) 2012 Nacho Mas. By default Standard firmata write analog
134   input value to PWM pin directly and send the ADC readings to analog output
135   without modification. By this modification you can adapt this behaviour.
136
137   two functions are added to the original StandardFirmata.:
138
139   mapAndSendAnalog(int pin):
140
141   Change the value returned by readAnalog before send through
142   firmata protocol. By this you can adapt the 0-1024 stadard ADC range
143   to more apropiate range (i.e phisical range of a sensor). Also you
144   can do some logic or event sent a variable value instead of
145   readAnalog.
146
147   mapAndWriteAnalog(int pin,int value):
148
149   Change the value recived through firmata protocol before write to
150   PWM output. Alternative can be used to change internal variable value instead
151   of setting PWM output.
152
153   (TODO: same for digital input/output)
154
155   NOTE: This only a template and by default do nothing! You have to addapt
156   to your real needs before.
157
158   Copyright (C) 2006-2008 Hans-Christoph Steiner.  All rights reserved.
159   Copyright (C) 2010-2011 Paul Stoffregen.  All rights reserved.
160   Copyright (C) 2009 Shigeru Kobayashi.  All rights reserved.
161   Copyright (C) 2009-2011 Jeff Hoefs.  All rights reserved.
162
163   This library is free software; you can redistribute it and/or
164   modify it under the terms of the GNU Lesser General Public
165   License as published by the Free Software Foundation; either
166   version 2.1 of the License, or (at your option) any later version.
167
168   See file LICENSE.txt for further informations on licensing terms.
169
170   formatted using the GNU C formatting and indenting
171 */
172
173 /*
174    TODO: use Program Control to load stored profiles from EEPROM
175 */
176
177 #include <Servo.h>
178 #include "Wire.h"
```

```
179 #include <Firmata.h>
180
181 //code for dogm
182 #include <SPI.h>
183 #include <dogm_7036.h>
184 dogm_7036 DOG;
185 word sw=0;//switching counter for change display
186 int tf=1; //time factor
187 char DispOut[6];  //char Array convert float to char array
188
189 #ifdef USE_MLX_SENSOR
190 #include "Adafruit_MLX90614.h"
191 Adafruit_MLX90614 mlx = Adafruit_MLX90614();
192 #endif //USE_MLX_SENSOR
193
194 #ifdef USE_DHT_SENSOR
195 #include "DHT.h"
196
197 #define DHTPIN 3     // Digital pin connected to the DHT sensor
198
199 // Uncomment whatever type you're using!
200 //#define DHTTYPE DHT11   // DHT 11
201 #define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
202 //#define DHTTYPE DHT21   // DHT 21 (AM2301)
203
204 DHT dht(DHTPIN, DHTTYPE);
205 #endif //USE_DHT_SENSOR
206
207 #ifdef USE_P_SENSOR
208   #include "Adafruit_BMP085.h"
209   Adafruit_BMP085 bmp;
210 #endif //USE_P_SENSOR
211
212 #ifdef USE_BME_SENSOR
213 #include "Adafruit_BME280.h"
214 Adafruit_BME280 bme;
215 #endif //USE_BME_SENSOR
216
217 #ifdef USE_TSL_SENSOR
218 #include "Adafruit_TSL2591.h"
219 Adafruit_TSL2591 tsl = Adafruit_TSL2591();
220 #endif //USE_TSL_SENSOR
221
222
223 float P, HR, IR, T, Tp, Thr, Tir, Dew, Light, brightness, lux, mag_arcsec2, Clouds, skyT;
224 int cloudy, dewing, frezzing;
225 bool mlxSuccess, bmpSuccess, bmeSuccess, tslSuccess, dhtSuccess;
226
```

```
227 #define TOTAL_ANALOG_PINS       11
228 #define TOTAL_PINS              25
229
230 void setupMeteoStation( ) {
231 #ifdef USE_MLX_SENSOR
232   if (!(mlxSuccess = mlx.begin())) {
233     //set IR sensor fail flag
234     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_MLX), STATUS_NOT_OK);
235     IR = 0;
236     Tir = 0;
237   } else digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_MLX), STATUS_OK); //Make sure IR sensor fail flag is off on success
238
239 #endif //USE_MLX_SENSOR
240
241 #ifdef USE_TSL_SENSOR
242   if (!(tslSuccess = tsl.begin())) {
243     //set TSL sensor fail flag
244     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_TSL), STATUS_OK);
245     mag_arcsec2 = 0.0;
246   } else digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_TSL), STATUS_NOT_OK); //Make sure TSL sensor fail flag is off on success
247
248 #endif //USE_MLX_SENSOR
249
250 #ifdef USE_P_SENSOR
251         if (!(bmpSuccess=bmp.begin())) {
252             //set P sensor fail flag
253             digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_BMP), STATUS_NOT_OK);
254             Tp=0;
255             P=0;
256         } else digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_BMP), STATUS_OK); //Make sure P sensor fail flag is off on success
257
258 #endif //USE_P_SENSOR
259
260 #ifndef USE_IRRADIANCE_SENSOR
261     Light=0;
262 #endif //USE_IRRADIANCE_SENSOR
263
264 #ifdef USE_BME_SENSOR
265   if (!(bmeSuccess = bme.begin())) {
266     //set P sensor fail flag
267     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_BME), STATUS_NOT_OK);
268     Tp = 0;
269     P = 0;
270     HR = 0;
271   } else digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_BME), STATUS_OK); //Make sure P/BME sensor fail flag is off on success
272
273 #endif //USE_BME_SENSOR
274
```

```
#ifdef USE_DHT_SENSOR
 if (dhtSuccess == false) {
    dht.begin();
    // check if we really get a proper result to ensure
    // that the initialization succeeded
    dhtSuccess = !isnan(dht.readHumidity());
    if (dhtSuccess)
      digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_TSL), STATUS_OK);
    else
      digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_TSL), STATUS_NOT_OK);

  }
#endif //USE_DHT_SENSOR

}

/*=========================================================================
   METEOSTATION FUNCTIONS
  =========================================================================*/

#ifdef USE_TSL_SENSOR
void configureSensor(tsl2591Gain_t gainSetting, tsl2591IntegrationTime_t timeSetting)
{
  // You can change the gain on the fly, to adapt to brighter/dimmer light situations
  tsl.setGain(gainSetting);

  // Changing the integration time gives you a longer time over which to sense light
  // longer timelines are slower, but are good in very low light situtations!
  tsl.setTiming(timeSetting);

  /* Display the gain and integration time for reference sake */
  tsl2591Gain_t gain = tsl.getGain();
}

float advancedLightSensorRead(void)
{
  // More advanced data read example. Read 32 bits with top 16 bits IR, bottom 16 bits full spectrum
  // That way you can do whatever math and comparisons you want!
  uint32_t lum = tsl.getFullLuminosity();
  uint16_t ir, full;
  float lux;

//  Serial.print("advancedLightSensorRead: GAIN = "); Serial.println(tsl.getGain());
//  Serial.print("advancedLightSensorRead: INTEGRATIONTIME = "); Serial.println(100 + 100*tsl.getTiming());
  lum = tsl.getFullLuminosity(); // first reading will be incorrect of Gain or Time was changed
  ir = lum >> 16;
  full = lum & 0xFFFF;
//  Serial.print("advancedLightSensorRead: lum = "); Serial.println(lum);
```

```
323 //  Serial.print("advancedLightSensorRead: ir = "); Serial.println(ir);
324 //  Serial.print("advancedLightSensorRead: full = "); Serial.println(full);
325   lux = tsl.calculateLux(full, ir);
326 //  Serial.print("advancedLightSensorRead: lux = "); Serial.println(lux);
327
328   if (full < 100){ //Increase GAIN (and INTEGRATIONTIME) if light level too low
329     switch (tsl.getGain())
330     {
331     case TSL2591_GAIN_LOW :
332       configureSensor(TSL2591_GAIN_MED, TSL2591_INTEGRATIONTIME_200MS);
333       break;
334     case TSL2591_GAIN_MED :
335       configureSensor(TSL2591_GAIN_HIGH, TSL2591_INTEGRATIONTIME_200MS);
336       break;
337     case TSL2591_GAIN_HIGH :
338       configureSensor(TSL2591_GAIN_MAX, TSL2591_INTEGRATIONTIME_200MS);
339       break;
340     case TSL2591_GAIN_MAX :
341       if(full < 100) {
342         switch (tsl.getTiming())
343         {
344         case TSL2591_INTEGRATIONTIME_200MS :
345           configureSensor(TSL2591_GAIN_MAX, TSL2591_INTEGRATIONTIME_300MS);
346           break;
347         case TSL2591_INTEGRATIONTIME_300MS :
348           configureSensor(TSL2591_GAIN_MAX, TSL2591_INTEGRATIONTIME_400MS);
349           break;
350         case TSL2591_INTEGRATIONTIME_400MS :
351           configureSensor(TSL2591_GAIN_MAX, TSL2591_INTEGRATIONTIME_500MS);
352           break;
353         case TSL2591_INTEGRATIONTIME_500MS :
354           configureSensor(TSL2591_GAIN_MAX, TSL2591_INTEGRATIONTIME_600MS);
355           break;
356         default:
357           configureSensor(TSL2591_GAIN_MAX, TSL2591_INTEGRATIONTIME_600MS);
358           break;
359         }
360       }
361       break;
362     default:
363       configureSensor(TSL2591_GAIN_MED, TSL2591_INTEGRATIONTIME_200MS);
364       break;
365     }
366   }
367
368   if (full > 30000){ //Decrease GAIN (and INTEGRATIONTIME) if light level too high
369     switch (tsl.getGain())
370     {
```

```
371     case TSL2591_GAIN_LOW :
372       break;
373     case TSL2591_GAIN_MED :
374       configureSensor(TSL2591_GAIN_LOW, TSL2591_INTEGRATIONTIME_200MS);
375       break;
376     case TSL2591_GAIN_HIGH :
377       configureSensor(TSL2591_GAIN_MED, TSL2591_INTEGRATIONTIME_200MS);
378       break;
379     case TSL2591_GAIN_MAX :
380       configureSensor(TSL2591_GAIN_HIGH, TSL2591_INTEGRATIONTIME_200MS);
381       break;
382     default:
383       configureSensor(TSL2591_GAIN_MED, TSL2591_INTEGRATIONTIME_200MS);
384       break;
385     }
386   }
387   return lux;
388 }
389
390 float get_mag_arcsec2(float lux)
391 {
392 //  Serial.print("get_mag_arcsec2: GAIN = "); Serial.println(tsl.getGain());
393 //  Serial.print("get_mag_arcsec2: INTEGRATIONTIME = "); Serial.println(100 + 100*tsl.getTiming());
394 //  Serial.print("get_mag_arcsec2: lux = "); Serial.println(lux);
395
396   mag_arcsec2 = log10(lux/108000)/-0.4;  //(log((ulux/108000) ) /(-0.4)
397 //  Serial.print("get_mag_arcsec2: mag_arcsec2 = "); Serial.println(mag_arcsec2);
398
399   return(mag_arcsec2);
400 }
401 #endif
402
403 void runMeteoStation() {
404
405 #ifdef USE_MLX_SENSOR
406   if (mlxSuccess) {
407     Tir = mlx.readAmbientTempC() / 100.0;
408     IR = mlx.readObjectTempC() / 100.0;
409   } else if (mlxSuccess = mlx.begin()) {
410     // Retry mlx.begin(), and clear MLX sensor fail flag
411     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_MLX), STATUS_OK);
412   }
413
414   Clouds = cloudIndex();
415   skyT = skyTemp();
416   if (Clouds > CLOUD_FLAG_PERCENT) {
417     cloudy = IPS_ALERT;
418   } else {
```

```
419     cloudy = IPS_OK;
420   }
421 #else
422   //set MLX sensor fail flag
423   digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_MLX), STATUS_NOT_OK);
424 #endif //USE_MLX_SENSOR
425
426 #ifdef USE_DHT_SENSOR
427     HR=dht.readHumidity();
428     Thr=dht.readTemperature();
429
430     Dew=dewPoint(Thr,HR);
431     if (Thr<=Dew+2) {
432         dewing=IPS_ALERT;
433     } else {
434         dewing=IPS_OK;
435     }
436 #else
437   #ifndef USE_TSL_SENSOR
438     //set HR sensor fail flag
439     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_TSL), STATUS_NOT_OK);
440   #endif
441 #endif //USE_DHT_SENSOR
442
443 #ifdef USE_TSL_SENSOR
444   if (tslSuccess) {
445     brightness = advancedLightSensorRead();
446     mag_arcsec2 = get_mag_arcsec2(brightness);
447   } else if (tslSuccess = tsl.begin()) {
448     // Retry tsl.begin(), and clear TSL sensor fail flag
449     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_TSL), STATUS_OK);
450   }
451 #else
452   #ifndef USE_DHT_SENSOR
453     //set HR sensor fail flag
454     digitalWrite(PIN_TO_DIGITAL(8), STATUS_NOT_OK);
455   #endif
456 #endif //USE_TSL_SENSOR
457
458 #ifdef USE_P_SENSOR
459     if (bmpSuccess) {
460         Tp=bmp.readTemperature();
461         P=bmp.readPressure();
462     } else if (bmpSuccess=bmp.begin()) {
463         // Retry bmp.begin(), and clear P sensor fail flag
464         digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_BMP), STATUS_OK);
465     }
466 #else
```

```
467   #ifndef USE_BME_SENSOR
468     //set P sensor fail flag
469     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_BME), STATUS_NOT_OK);
470   #endif
471 #endif //USE_P_SENSOR
472
473 #ifdef USE_BME_SENSOR
474   if (bmeSuccess) {
475     Tp = bme.readTemperature();
476     P = bme.readPressure();
477     HR = bme.readHumidity();
478
479     Dew = dewPoint(Tp, HR);
480     if (Tp <= Dew + 2) {
481       dewing = 1;
482     } else {
483       dewing = 0;
484     }
485   } else if (bmeSuccess = bme.begin()) {
486     // Retry bme.begin(), and clear BME sensor fail flag
487     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_BME), STATUS_OK);
488   }
489 #else
490   #ifndef USE_P_SENSOR
491     //set BME sensor fail flag
492     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_BME), STATUS_NOT_OK);
493   #endif
494 #endif //USE_BME_SENSOR
495
496 #ifdef USE_IRRADIANCE_SENSOR
497     Light=analogRead(IR_RADIANCE_PIN);
498 #endif //USE_IRRADIANCE_SENSOR
499
500 #if defined T_MAIN_Thr
501     T=Thr;
502 #elif defined T_MAIN_Tir
503   T = Tir;
504 #elif defined T_MAIN_Tp
505   T = Tp;
506 #endif  //T_MAIN
507
508     if (T<FREZZING) {
509       frezzing=IPS_ALERT;
510     } else {
511       frezzing=IPS_OK;
512     }
513 }
514
```

```
515 void checkMeteo() {
516
517     if (cloudy==IPS_OK) {
518        digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_CLOUDY), STATUS_OK); // enable internal pull-ups
519     } else {
520        digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_CLOUDY), STATUS_NOT_OK); // disable internal pull-ups
521     }
522
523     if (dewing==IPS_OK) {
524        digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_DEW), STATUS_OK); // enable internal pull-ups
525     } else {
526        digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_DEW), STATUS_NOT_OK); // disable internal pull-ups
527     }
528
529     if (frezzing==IPS_OK) {
530        digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_FREZZY), STATUS_OK); // enable internal pull-ups
531     } else {
532        digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_FREZZY), STATUS_NOT_OK); // disable internal pull-ups
533     }
534
535 #ifdef USE_IRRADIANCE_SENSOR
536     if (Light <= MINIMUM_DAYLIGHT) {
537        digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_DAYLIGHT), STATUS_OK); // enable internal pull-ups
538     } else {
539        digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_DAYLIGHT), STATUS_NOT_OK); // disable internal pull-ups
540     }
541 #endif //USE_IRRADIANCE_SENSOR
542
543 #ifdef USE_TSL_SENSOR
544   if (mag_arcsec2 > DAYLIGHT) {
545     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_SQM), STATUS_OK); // enable internal pull-ups
546   } else {
547     digitalWrite(PIN_TO_DIGITAL(PIN_STATUS_SQM), STATUS_NOT_OK); // disable internal pull-ups
548   }
549 #endif //USE_TSL_SENSOR
550
551 }
552
553 // dewPoint function NOAA
554 // reference: http://wahiduddin.net/calc/density_algorithms.htm
555 double dewPoint(double celsius, double humidity)
556 {
557   double A0 = 373.15 / (273.15 + celsius);
558   double SUM = -7.90298 * (A0 - 1);
559   SUM += 5.02808 * log10(A0);
560   SUM += -1.3816e-7 * (pow(10, (11.344 * (1 - 1 / A0))) - 1);
561   SUM += 8.1328e-3 * (pow(10, (-3.49149 * (A0 - 1))) - 1);
562   SUM += log10(1013.246);
```

```
563   double VP = pow(10, SUM - 3) * humidity;
564   double T = log(VP / 0.61078); // temp var
565   return (241.88 * T) / (17.558 - T);
566 }
567
568 // delta max = 0.6544 wrt dewPoint()
569 // 5x faster than dewPoint()
570 // reference: http://en.wikipedia.org/wiki/Dew_point
571 double dewPointFast(double celsius, double humidity)
572 {
573   double a = 17.271;
574   double b = 237.7;
575   double temp = (a * celsius) / (b + celsius) + log(humidity / 100);
576   double Td = (b * temp) / (a - temp);
577   return Td;
578 }
579
580 #ifdef USE_MLX_SENSOR
581 //From AAG Cloudwatcher formula. Need to improve futher.
582 //http://www.aagware.eu/aag/cloudwatcher700/WebHelp/index.htm#page=Operational%20Aspects/23-TemperatureFactor-.htm
583 //https://azug.minpet.unibas.ch/wikiobsvermes/index.php/AAG_cloud_sensor#Snow_on_the_sky_temperature_sensor
584 double skyTemp() {
585   //Constant defined above
586   double Td = (K1 / 100.) * (T - K2 / 10) + (K3 / 100.) * pow((exp (K4 / 1000.* T)) , (K5 / 100.));
587   double Tsky = IR - Td;
588   return Tsky;
589 }
590
591
592 double cloudIndex() {
593   double Tcloudy = CLOUD_TEMP_OVERCAST, Tclear = CLOUD_TEMP_CLEAR;
594   double Tsky = skyTemp();
595   double Index;
596   if (Tsky < Tclear) Tsky = Tclear;
597   if (Tsky > Tcloudy) Tsky = Tcloudy;
598   Index = (Tsky - Tclear) * 100 / (Tcloudy - Tclear);
599   return Index;
600 }
601 #endif //USE_MLX_SENSOR
602
603 /* Nacho Mas.
604   Change the value returned by readAnalog before send through
605   firmata protocol. By this you can adapt the 0-1024 stadard ADC range
606   to more apropiate range (i.e phisical range of a sensor). Also you
607   can do some logic or event sent a variable value instead of
608   readAnalog.
609 */
610 int mapAndSendAnalog(int pin) {
```

```
611
612   //some scalation are use. Don't change without changing also skeleton file
613   int value = 0;
614   int result = 0;
615
616   switch (pin) {
617     //PIN 14->A0, 24->A10
618     case 0:
619       result = (IR + 273) * 20;
620       break;
621     case 1:
622       result = (Tir + 273) * 20;
623       break;
624     case 2:
625       result = (P / 10);
626       break;
627     case 3:
628       result = (Tp + 273) * 20;
629       break;
630     case 4:
631       result = HR * 100;
632       break;
633     case 5:
634       result = (Thr + 273) * 20;
635       break;
636     case 6:
637       result = (Dew + 273) * 20;
638       break;
639     case 7:
640 #ifdef USE_TSL_SENSOR
641       result = mag_arcsec2 * 100;
642 #endif
643 #ifdef USE_IRRADIANCE_SENSOR
644       result=Light;
645 #endif
646       break;
647     case 8:
648       result = Clouds;
649       break;
650     case 9:
651       result = (skyT + 273) * 20;
652       break;
653     case 10:
654       result = (T + 273) * 20;
655       break;
656
657
658     default:
```

```
659       result = value;
660       break;
661   }
662   Firmata.sendAnalog(pin, result);
663 }
664
665 /* Nacho Mas.
666   Change the value recived through firmata protocol before write to
667   PWM output. Alternative can be used to change internal variable value instead
668   of setting PWM output.
669 */
670 int mapAndWriteAnalog(int pin, int value) {
671   int pwmPin = PIN_TO_PWM(pin);
672   int result = 0;
673   switch (pwmPin) {
674     case 5:
675     case 6:
676     case 9:
677     //             result=map(value, 0, 100, 0, 255);
678     //             break;
679     default:
680       result = value;
681       break;
682   }
683   analogWrite(pwmPin, result);
684 }
685
686 /*==============================================================================
687    STANDAR FIRMATA FUNCTIONS
688   ==============================================================================*/
689
690 void disableI2CPins();
691 void enableI2CPins();
692 void reportAnalogCallback(byte analogPin, int value);
693
694 // move the following defines to Firmata.h?
695 #define I2C_WRITE B00000000
696 #define I2C_READ B00001000
697 #define I2C_READ_CONTINUOUSLY B00010000
698 #define I2C_STOP_READING B00011000
699 #define I2C_READ_WRITE_MODE_MASK B00011000
700 #define I2C_10BIT_ADDRESS_MODE_MASK B00100000
701
702 #define MAX_QUERIES 1
703 #define MINIMUM_SAMPLING_INTERVAL 1000 //
704
705 #define REGISTER_NOT_SPECIFIED -1
706
```

```
707 /*==============================================================================
708    GLOBAL VARIABLES
709   ==============================================================================*/
710
711 /* analog inputs */
712 int analogInputsToReport = 0; // bitwise array to store pin reporting
713
714 /* digital input ports */
715 byte reportPINs[TOTAL_PORTS];       // 1 = report this port, 0 = silence
716 byte previousPINs[TOTAL_PORTS];     // previous 8 bits sent
717
718 /* pins configuration */
719 byte pinConfig[TOTAL_PINS];         // configuration of every pin
720 byte portConfigInputs[TOTAL_PORTS]; // each bit: 1 = pin in INPUT, 0 = anything else
721 int pinState[TOTAL_PINS];           // any value that has been written
722
723 /* timer variables */
724 unsigned long currentMillis;        // store the current value from millis()
725 unsigned long previousMillis = 0;     // for comparison with currentMillis
726 int samplingInterval = 19;          // how often to run the main loop (in ms)
727
728 /* i2c data */
729 struct i2c_device_info {
730   byte addr;
731   byte reg;
732   byte bytes;
733 };
734
735 /* for i2c read continuous more */
736 i2c_device_info query[MAX_QUERIES];
737
738 byte i2cRxData[32];
739 boolean isI2CEnabled = false;
740 signed char queryIndex = -1;
741 unsigned int i2cReadDelayTime = 0;  // default delay time between i2c read request and Wire.requestFrom()
742
743 Servo servos[MAX_SERVOS];
744
745 void readAndReportData(byte address, int theRegister, byte numBytes) {
746   // allow I2C requests that don't require a register read
747   // for example, some devices using an interrupt pin to signify new data available
748   // do not always require the register read so upon interrupt you call Wire.requestFrom()
749   if (theRegister != REGISTER_NOT_SPECIFIED) {
750     Wire.beginTransmission(address);
751 #if ARDUINO >= 100
752     Wire.write((byte)theRegister);
753 #else
754     Wire.send((byte)theRegister);
```

```
755 #endif
756     Wire.endTransmission();
757     delayMicroseconds(i2cReadDelayTime);  // delay is necessary for some devices such as WiiNunchuck
758   } else {
759     theRegister = 0;  // fill the register with a dummy value
760   }
761
762   Wire.requestFrom(address, numBytes);  // all bytes are returned in requestFrom
763
764   // check to be sure correct number of bytes were returned by slave
765   if (numBytes == Wire.available()) {
766     i2cRxData[0] = address;
767     i2cRxData[1] = theRegister;
768     for (int i = 0; i < numBytes; i++) {
769 #if ARDUINO >= 100
770       i2cRxData[2 + i] = Wire.read();
771 #else
772       i2cRxData[2 + i] = Wire.receive();
773 #endif
774     }
775   }
776   else {
777     if (numBytes > Wire.available()) {
778       Firmata.sendString("I2C Read Error: Too many bytes received");
779     } else {
780       Firmata.sendString("I2C Read Error: Too few bytes received");
781     }
782   }
783
784   // send slave address, register and received bytes
785   Firmata.sendSysex(SYSEX_I2C_REPLY, numBytes + 2, i2cRxData);
786 }
787
788 void outputPort(byte portNumber, byte portValue, byte forceSend)
789 {
790   // pins not configured as INPUT are cleared to zeros
791   portValue = portValue & portConfigInputs[portNumber];
792   // only send if the value is different than previously sent
793   if (forceSend || previousPINs[portNumber] != portValue) {
794     Firmata.sendDigitalPort(portNumber, portValue);
795     previousPINs[portNumber] = portValue;
796   }
797 }
798
799 /* -----------------------------------------------------------------------------
800    check all the active digital inputs for change of state, then add any events
801    to the Serial output queue using Serial.print() */
802 void checkDigitalInputs(void)
```

```
803 {
804   /* Using non-looping code allows constants to be given to readPort().
805      The compiler will apply substantial optimizations if the inputs
806      to readPort() are compile-time constants. */
807   //Nacho Mas.
808   //TODO: Cach deafult behaviour
809   boolean send_always = false;
810   if (TOTAL_PORTS > 0 && reportPINs[0]) outputPort(0, readPort(0, portConfigInputs[0]), send_always);
811   if (TOTAL_PORTS > 1 && reportPINs[1]) outputPort(1, readPort(1, portConfigInputs[1]), send_always);
812   if (TOTAL_PORTS > 2 && reportPINs[2]) outputPort(2, readPort(2, portConfigInputs[2]), send_always);
813   if (TOTAL_PORTS > 3 && reportPINs[3]) outputPort(3, readPort(3, portConfigInputs[3]), send_always);
814   if (TOTAL_PORTS > 4 && reportPINs[4]) outputPort(4, readPort(4, portConfigInputs[4]), send_always);
815   if (TOTAL_PORTS > 5 && reportPINs[5]) outputPort(5, readPort(5, portConfigInputs[5]), send_always);
816   if (TOTAL_PORTS > 6 && reportPINs[6]) outputPort(6, readPort(6, portConfigInputs[6]), send_always);
817   if (TOTAL_PORTS > 7 && reportPINs[7]) outputPort(7, readPort(7, portConfigInputs[7]), send_always);
818   if (TOTAL_PORTS > 8 && reportPINs[8]) outputPort(8, readPort(8, portConfigInputs[8]), send_always);
819   if (TOTAL_PORTS > 9 && reportPINs[9]) outputPort(9, readPort(9, portConfigInputs[9]), send_always);
820   if (TOTAL_PORTS > 10 && reportPINs[10]) outputPort(10, readPort(10, portConfigInputs[10]), send_always);
821   if (TOTAL_PORTS > 11 && reportPINs[11]) outputPort(11, readPort(11, portConfigInputs[11]), send_always);
822   if (TOTAL_PORTS > 12 && reportPINs[12]) outputPort(12, readPort(12, portConfigInputs[12]), send_always);
823   if (TOTAL_PORTS > 13 && reportPINs[13]) outputPort(13, readPort(13, portConfigInputs[13]), send_always);
824   if (TOTAL_PORTS > 14 && reportPINs[14]) outputPort(14, readPort(14, portConfigInputs[14]), send_always);
825   if (TOTAL_PORTS > 15 && reportPINs[15]) outputPort(15, readPort(15, portConfigInputs[15]), send_always);
826 }
827
828 // -----------------------------------------------------------------------------
829 /* sets the pin mode to the correct state and sets the relevant bits in the
830    two bit-arrays that track Digital I/O and PWM status
831 */
832 void setPinModeCallback(byte pin, int mode)
833 {
834   if (pinConfig[pin] == I2C && isI2CEnabled && mode != I2C) {
835     // disable i2c so pins can be used for other functions
836     // the following if statements should reconfigure the pins properly
837     disableI2CPins();
838   }
839   if (IS_PIN_SERVO(pin) && mode != SERVO && servos[PIN_TO_SERVO(pin)].attached()) {
840     servos[PIN_TO_SERVO(pin)].detach();
841   }
842   if (IS_PIN_ANALOG(pin)) {
843     reportAnalogCallback(PIN_TO_ANALOG(pin), mode == ANALOG ? 1 : 0); // turn on/off reporting
844   }
845   if (IS_PIN_DIGITAL(pin)) {
846     if (mode == INPUT) {
847       portConfigInputs[pin / 8] |= (1 << (pin & 7));
848     } else {
849       portConfigInputs[pin / 8] &= ~(1 << (pin & 7));
850     }
```

```
851   }
852   pinState[pin] = 0;
853   switch (mode) {
854     case ANALOG:
855       if (IS_PIN_ANALOG(pin)) {
856         if (IS_PIN_DIGITAL(pin)) {
857           pinMode(PIN_TO_DIGITAL(pin), INPUT); // disable output driver
858           digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
859         }
860         pinConfig[pin] = ANALOG;
861       }
862       break;
863     case INPUT:
864       if (IS_PIN_DIGITAL(pin)) {
865         pinMode(PIN_TO_DIGITAL(pin), INPUT); // disable output driver
866         digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
867         pinConfig[pin] = INPUT;
868       }
869       break;
870     case OUTPUT:
871       if (IS_PIN_DIGITAL(pin)) {
872         digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable PWM
873         pinMode(PIN_TO_DIGITAL(pin), OUTPUT);
874         pinConfig[pin] = OUTPUT;
875       }
876       break;
877     case PWM:
878       if (IS_PIN_PWM(pin)) {
879         pinMode(PIN_TO_PWM(pin), OUTPUT);
880         analogWrite(PIN_TO_PWM(pin), 0);
881         pinConfig[pin] = PWM;
882       }
883       break;
884     case SERVO:
885       if (IS_PIN_SERVO(pin)) {
886         pinConfig[pin] = SERVO;
887         if (!servos[PIN_TO_SERVO(pin)].attached()) {
888           servos[PIN_TO_SERVO(pin)].attach(PIN_TO_DIGITAL(pin));
889         }
890       }
891       break;
892     case I2C:
893       if (IS_PIN_I2C(pin)) {
894         // mark the pin as i2c
895         // the user must call I2C_CONFIG to enable I2C for a device
896         pinConfig[pin] = I2C;
897       }
898       break;
```

```
899     default:
900       Firmata.sendString("Unknown pin mode"); // TODO: put error msgs in EEPROM
901   }
902   // TODO: save status to EEPROM here, if changed
903 }
904
905 void analogWriteCallback(byte pin, int value)
906 {
907   if (pin < TOTAL_PINS) {
908     switch (pinConfig[pin]) {
909       case SERVO:
910         if (IS_PIN_SERVO(pin))
911           servos[PIN_TO_SERVO(pin)].write(value);
912         pinState[pin] = value;
913         break;
914       case PWM:
915         if (IS_PIN_PWM(pin))
916           //Nacho Mas. Write analog and do something before set PWM.
917           mapAndWriteAnalog(pin, value);
918         pinState[pin] = value;
919         break;
920     }
921   }
922 }
923
924 void digitalWriteCallback(byte port, int value)
925 {
926   byte pin, lastPin, mask = 1, pinWriteMask = 0;
927
928   if (port < TOTAL_PORTS) {
929     // create a mask of the pins on this port that are writable.
930     lastPin = port * 8 + 8;
931     if (lastPin > TOTAL_PINS) lastPin = TOTAL_PINS;
932     for (pin = port * 8; pin < lastPin; pin++) {
933       // do not disturb non-digital pins (eg, Rx & Tx)
934       if (IS_PIN_DIGITAL(pin)) {
935         // only write to OUTPUT and INPUT (enables pullup)
936         // do not touch pins in PWM, ANALOG, SERVO or other modes
937         if (pinConfig[pin] == OUTPUT || pinConfig[pin] == INPUT) {
938           pinWriteMask |= mask;
939           pinState[pin] = ((byte)value & mask) ? 1 : 0;
940         }
941       }
942       mask = mask << 1;
943     }
944     //Nacho Mas. TODO: sustitute this call by
945     //mapAndWriteDigital(port, (byte)value, pinWriteMask)
946     writePort(port, (byte)value, pinWriteMask);
```

```
947   }
948 }
949
950
951 // ------------------------------------------------------------------------
952 /* sets bits in a bit array (int) to toggle the reporting of the analogIns
953 */
954 //void FirmataClass::setAnalogPinReporting(byte pin, byte state) {
955 //}
956 void reportAnalogCallback(byte analogPin, int value)
957 {
958   if (analogPin < TOTAL_ANALOG_PINS) {
959     if (value == 0) {
960       analogInputsToReport = analogInputsToReport & ~ (1 << analogPin);
961     } else {
962       analogInputsToReport = analogInputsToReport | (1 << analogPin);
963     }
964   }
965   // TODO: save status to EEPROM here, if changed
966 }
967
968 void reportDigitalCallback(byte port, int value)
969 {
970   if (port < TOTAL_PORTS) {
971     reportPINs[port] = (byte)value;
972   }
973   // do not disable analog reporting on these 8 pins, to allow some
974   // pins used for digital, others analog.  Instead, allow both types
975   // of reporting to be enabled, but check if the pin is configured
976   // as analog when sampling the analog inputs.  Likewise, while
977   // scanning digital pins, portConfigInputs will mask off values from any
978   // pins configured as analog
979 }
980
981 /*==============================================================================
982    SYSEX-BASED commands
983   ============================================================================*/
984
985 void sysexCallback(byte command, byte argc, byte *argv)
986 {
987   byte mode;
988   byte slaveAddress;
989   byte slaveRegister;
990   byte data;
991   unsigned int delayTime;
992
993   switch (command) {
994     case I2C_REQUEST:
```

```
995       mode = argv[1] & I2C_READ_WRITE_MODE_MASK;
996       if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {
997         Firmata.sendString("10-bit addressing mode is not yet supported");
998         return;
999       }
1000      else {
1001        slaveAddress = argv[0];
1002      }
1003
1004      switch (mode) {
1005        case I2C_WRITE:
1006          Wire.beginTransmission(slaveAddress);
1007          for (byte i = 2; i < argc; i += 2) {
1008            data = argv[i] + (argv[i + 1] << 7);
1009 #if ARDUINO >= 100
1010            Wire.write(data);
1011 #else
1012            Wire.send(data);
1013 #endif
1014          }
1015          Wire.endTransmission();
1016          delayMicroseconds(70);
1017          break;
1018        case I2C_READ:
1019          if (argc == 6) {
1020            // a slave register is specified
1021            slaveRegister = argv[2] + (argv[3] << 7);
1022            data = argv[4] + (argv[5] << 7);  // bytes to read
1023            readAndReportData(slaveAddress, (int)slaveRegister, data);
1024          }
1025          else {
1026            // a slave register is NOT specified
1027            data = argv[2] + (argv[3] << 7);  // bytes to read
1028            readAndReportData(slaveAddress, (int)REGISTER_NOT_SPECIFIED, data);
1029          }
1030          break;
1031        case I2C_READ_CONTINUOUSLY:
1032          if ((queryIndex + 1) >= MAX_QUERIES) {
1033            // too many queries, just ignore
1034            Firmata.sendString("too many queries");
1035            break;
1036          }
1037          queryIndex++;
1038          query[queryIndex].addr = slaveAddress;
1039          query[queryIndex].reg = argv[2] + (argv[3] << 7);
1040          query[queryIndex].bytes = argv[4] + (argv[5] << 7);
1041          break;
1042        case I2C_STOP_READING:
```

```
1043          byte queryIndexToSkip;
1044          // if read continuous mode is enabled for only 1 i2c device, disable
1045          // read continuous reporting for that device
1046          if (queryIndex <= 0) {
1047            queryIndex = -1;
1048          } else {
1049            // if read continuous mode is enabled for multiple devices,
1050            // determine which device to stop reading and remove it's data from
1051            // the array, shifiting other array data to fill the space
1052            for (byte i = 0; i < queryIndex + 1; i++) {
1053              if (query[i].addr = slaveAddress) {
1054                queryIndexToSkip = i;
1055                break;
1056              }
1057            }
1058
1059            for (byte i = queryIndexToSkip; i < queryIndex + 1; i++) {
1060              if (i < MAX_QUERIES) {
1061                query[i].addr = query[i + 1].addr;
1062                query[i].reg = query[i + 1].addr;
1063                query[i].bytes = query[i + 1].bytes;
1064              }
1065            }
1066            queryIndex--;
1067          }
1068          break;
1069        default:
1070          break;
1071      }
1072      break;
1073    case I2C_CONFIG:
1074      delayTime = (argv[0] + (argv[1] << 7));
1075
1076      if (delayTime > 0) {
1077        i2cReadDelayTime = delayTime;
1078      }
1079
1080      if (!isI2CEnabled) {
1081        enableI2CPins();
1082      }
1083
1084      break;
1085    case SERVO_CONFIG:
1086      if (argc > 4) {
1087        // these vars are here for clarity, they'll optimized away by the compiler
1088        byte pin = argv[0];
1089        int minPulse = argv[1] + (argv[2] << 7);
1090        int maxPulse = argv[3] + (argv[4] << 7);
```

```
1091
1092          if (IS_PIN_SERVO(pin)) {
1093            if (servos[PIN_TO_SERVO(pin)].attached())
1094              servos[PIN_TO_SERVO(pin)].detach();
1095            servos[PIN_TO_SERVO(pin)].attach(PIN_TO_DIGITAL(pin), minPulse, maxPulse);
1096            setPinModeCallback(pin, SERVO);
1097          }
1098        }
1099        break;
1100      case SAMPLING_INTERVAL:
1101        if (argc > 1) {
1102          samplingInterval = argv[0] + (argv[1] << 7);
1103          if (samplingInterval < MINIMUM_SAMPLING_INTERVAL) {
1104            samplingInterval = MINIMUM_SAMPLING_INTERVAL;
1105          }
1106        } else {
1107          //Firmata.sendString("Not enough data");
1108        }
1109        break;
1110      case EXTENDED_ANALOG:
1111        if (argc > 1) {
1112          int val = argv[1];
1113          if (argc > 2) val |= (argv[2] << 7);
1114          if (argc > 3) val |= (argv[3] << 14);
1115          analogWriteCallback(argv[0], val);
1116        }
1117        break;
1118      case CAPABILITY_QUERY:
1119        tf=15;  //Time Factor *15
1120        Serial.write(START_SYSEX);
1121        Serial.write(CAPABILITY_RESPONSE);
1122        for (byte pin = 0; pin < TOTAL_PINS; pin++) {
1123          if (IS_PIN_DIGITAL(pin)) {
1124            Serial.write((byte)INPUT);
1125            Serial.write(1);
1126            Serial.write((byte)OUTPUT);
1127            Serial.write(1);
1128          }
1129          if (IS_PIN_ANALOG(pin)) {
1130            Serial.write(ANALOG);
1131            Serial.write(10);
1132          }
1133          if (IS_PIN_PWM(pin)) {
1134            Serial.write(PWM);
1135            Serial.write(8);
1136          }
1137          if (IS_PIN_SERVO(pin)) {
1138            Serial.write(SERVO);
```

```
1139          Serial.write(14);
1140        }
1141        if (IS_PIN_I2C(pin)) {
1142          Serial.write(I2C);
1143          Serial.write(1);  // to do: determine appropriate value
1144        }
1145        Serial.write(127);
1146      }
1147      Serial.write(END_SYSEX);
1148      break;
1149    case PIN_STATE_QUERY:
1150      if (argc > 0) {
1151        byte pin = argv[0];
1152        Serial.write(START_SYSEX);
1153        Serial.write(PIN_STATE_RESPONSE);
1154        Serial.write(pin);
1155        if (pin < TOTAL_PINS) {
1156          Serial.write((byte)pinConfig[pin]);
1157          Serial.write((byte)pinState[pin] & 0x7F);
1158          if (pinState[pin] & 0xFF80) Serial.write((byte)(pinState[pin] >> 7) & 0x7F);
1159          if (pinState[pin] & 0xC000) Serial.write((byte)(pinState[pin] >> 14) & 0x7F);
1160        }
1161        Serial.write(END_SYSEX);
1162      }
1163      break;
1164    case ANALOG_MAPPING_QUERY:
1165      Serial.write(START_SYSEX);
1166      Serial.write(ANALOG_MAPPING_RESPONSE);
1167      for (byte pin = 0; pin < TOTAL_PINS; pin++) {
1168        Serial.write(IS_PIN_ANALOG(pin) ? PIN_TO_ANALOG(pin) : 127);
1169      }
1170      Serial.write(END_SYSEX);
1171      break;
1172  }
1173 }
1174
1175 void enableI2CPins()
1176 {
1177   byte i;
1178   // is there a faster way to do this? would probaby require importing
1179   // Arduino.h to get SCL and SDA pins
1180   for (i = 0; i < TOTAL_PINS; i++) {
1181     if (IS_PIN_I2C(i)) {
1182       // mark pins as i2c so they are ignore in non i2c data requests
1183       setPinModeCallback(i, I2C);
1184     }
1185   }
1186
```

```
1187   isI2CEnabled = true;
1188
1189   // is there enough time before the first I2C request to call this here?
1190   Wire.begin();
1191 }
1192
1193 /* disable the i2c pins so they can be used for other functions */
1194 void disableI2CPins() {
1195   isI2CEnabled = false;
1196   // disable read continuous mode for all devices
1197   queryIndex = -1;
1198   // uncomment the following if or when the end() method is added to Wire library
1199   // Wire.end();
1200 }
1201
1202 /*==============================================================================
1203    SETUP()
1204   ============================================================================*/
1205
1206 void systemResetCallback()
1207 {
1208   // initialize a defalt state
1209   // TODO: option to load config from EEPROM instead of default
1210   if (isI2CEnabled) {
1211     disableI2CPins();
1212   }
1213   for (byte i = 0; i < TOTAL_PORTS; i++) {
1214     reportPINs[i] = false;       // by default, reporting off
1215     portConfigInputs[i] = 0;  // until activated
1216     previousPINs[i] = 0;
1217   }
1218   // pins with analog capability default to analog input
1219   // otherwise, pins default to digital output
1220   for (byte i = 0; i < TOTAL_PINS; i++) {
1221     if (IS_PIN_ANALOG(i)) {
1222       // turns off pullup, configures everything
1223       setPinModeCallback(i, ANALOG);
1224     } else {
1225       // sets the output to 0, configures portConfigInputs
1226       setPinModeCallback(i, OUTPUT);
1227     }
1228   }
1229   // by default, do not report any analog inputs
1230   analogInputsToReport = 0;
1231
1232   /* send digital inputs to set the initial state on the host computer,
1233      since once in the loop(), this firmware will only send on change */
1234   /*
```

```
1235      TODO: this can never execute, since no pins default to digital input
1236          but it will be needed when/if we support EEPROM stored config
1237      for (byte i=0; i < TOTAL_PORTS; i++) {
1238      outputPort(i, readPort(i, portConfigInputs[i]), true);
1239      }
1240    */
1241    setupMeteoStation();
1242  }
1243
1244  void dispinit(){   //init Display
1245    DOG.initialize(10,0,0,9,8,1,DOGM163);   //SS = 10, 0,0= use Hardware SPI, 9 = RS, 8 = RESET, 1 = 5V, EA DOGM163-A (=3 lines)
1246    DOG.displ_onoff(true);          //turn Display on
1247    DOG.cursor_onoff(false);         //turn Curosor blinking off
1248    DOG.clear_display();
1249    DOG.contrast(11); //162 - 35, 163 - 11
1250  }
1251
1252  void flout(char lb[8],float flch,int e1,int e2,int row){  //label[8] as string, value, unity char 1 and 2 in decimal, row 1-3
1253    dtostrf(flch, 6, 1,DispOut);
1254    DOG.position(1,row);  //column 1 output Label
1255    DOG.string(lb);
1256    DOG.position(9,row);  //column 9 output Value
1257    DOG.string(DispOut);
1258    DOG.position(15,row);  //column 15 output unity
1259    DOG.ascii(e1);
1260    DOG.ascii(e2);
1261  }
1262
1263  void dispside(int si){    //output side number as an small screen saver
1264    DOG.clear_display();
1265    DOG.position(5,2);
1266    DOG.string("Side ");
1267    DOG.ascii(si);
1268  }
1269
1270  void display(){     //output Display
1271  //skyT, mag_arcsec2,, T, P, HR,, Dew, Clouds,
1272    if (sw==30*tf) dispside(49);
1273    else if (sw==36*tf){
1274        dispinit();
1275        flout("SkyTemp",skyT,223,67,1);
1276        flout("SkyQual",mag_arcsec2,109,97,2);
1277        DOG.position(3,3);
1278        DOG.string("(mag/arcsec^2)");
1279    }
1280    else if (sw==60*tf) dispside(50);
1281    else if (sw==66*tf){
1282        DOG.clear_display();
```

```
1283        flout("Temper.",T,223,67,1);
1284        flout("Pressur",P/100,109,98,2);
1285        flout("Humid-R",HR,37,32,3);
1286
1287    }
1288    else if (sw==90*tf) dispside(51);
1289    else if (sw==96*tf) {
1290        DOG.clear_display();
1291        flout("Dewpoin",Dew,223,67,1);
1292        flout("Clouds",Clouds,37,32,3);
1293        sw = 0;
1294    }
1295    sw++;
1296 }
1297
1298 void setup()
1299 {
1300    Firmata.setFirmwareVersion(FIRMATA_MAJOR_VERSION, FIRMATA_MINOR_VERSION);
1301
1302    Firmata.attach(ANALOG_MESSAGE, analogWriteCallback);
1303    Firmata.attach(DIGITAL_MESSAGE, digitalWriteCallback);
1304    Firmata.attach(REPORT_ANALOG, reportAnalogCallback);
1305    Firmata.attach(REPORT_DIGITAL, reportDigitalCallback);
1306    Firmata.attach(SET_PIN_MODE, setPinModeCallback);
1307    Firmata.attach(START_SYSEX, sysexCallback);
1308    Firmata.attach(SYSTEM_RESET, systemResetCallback);
1309
1310    Firmata.begin(9600);
1311    systemResetCallback();  // reset to default config
1312
1313 //Init dogm
1314    dispinit();
1315    DOG.position(4,1);
1316    DOG.string("Welcome to");
1317    DOG.position(3,2);
1318    DOG.string("MeteoStation");
1319    delay(3000);
1320    DOG.clear_display();
1321    DOG.position(1,1);
1322    DOG.string("start the");
1323    DOG.position(1,2);
1324    DOG.string("measurements ..");
1325 }
1326
1327 /*=============================================================================
1328    LOOP()
1329    =============================================================================*/
1330 void loop()
```

```
1331 {
1332   byte pin, analogPin;
1333
1334   /* DIGITALREAD - as fast as possible, check for changes and output them to the
1335      FTDI buffer using Serial.print()  */
1336   //Nacho Mas. TODO: sustitute this call by
1337   //mapAndSendDigital()
1338   //checkDigitalInputs();
1339
1340   /* SERIALREAD - processing incoming messagse as soon as possible, while still
1341      checking digital inputs.  */
1342   while (Firmata.available())
1343     Firmata.processInput();
1344
1345   /* SEND FTDI WRITE BUFFER - make sure that the FTDI buffer doesn't go over
1346      60 bytes. use a timer to sending an event character every 4 ms to
1347      trigger the buffer to dump. */
1348
1349   currentMillis = millis();
1350   if (currentMillis - previousMillis > samplingInterval) {
1351     previousMillis += samplingInterval;
1352     checkMeteo();
1353     runMeteoStation();
1354     checkDigitalInputs();
1355     /* ANALOGREAD - do all analogReads() at the configured sampling interval */
1356     for (pin = 0; pin < TOTAL_PINS; pin++) {
1357       if (IS_PIN_ANALOG(pin) && pinConfig[pin] == ANALOG) {
1358         analogPin = PIN_TO_ANALOG(pin);
1359         if (analogInputsToReport & (1 << analogPin)) {
1360           //Nacho Mas. Read analog and do something before send. Then send it
1361           mapAndSendAnalog(analogPin);
1362         }
1363       }
1364     }
1365     // report i2c data for all device with read continuous mode enabled
1366     if (queryIndex > -1) {
1367       for (byte i = 0; i < queryIndex + 1; i++) {
1368         readAndReportData(query[i].addr, query[i].reg, query[i].bytes);
1369       }
1370     }
1371   }
1372   display();
1373   delay(10);    //time is needed by display
1374 }
```