# Project: Teensy Climate

The primary purpose of this project was to give me a reason to learn how to develop solutions using the Atmel AVR 8-bit microcontrollers.

The secondary purpose of this project is to develop a climate control system for the home.

# Current Project Status

4 Dec 2014:

Really haven't done anything with this project since the last update primarily because I wasn't finding a decent relay module for the fan attic fan controls easily enough. Earlier this week someone showed me the SainSmart relay modules. They just came in today. Hopefully I'll get to play with soon and get them integrated into the project. The ultimate goal of this project was to have a device automatically control the attic fan during the sprint, summer and autumn months. With these boards I now see a light at the end of the tunnel.

30 July 2011:

The code has been cleaned up a little bit (you should have seen versions 0.1-0.4!!!).

While I am using the uthash library, I know I'm not using it properly, but it works. While I wanted to implement a hash table, it's pretty much just creating a linked list for me and I'm iterating the list in various places. I've tried implementing a standard linked list using pointers (in version 0.6) but it's not working properly. The sensor objects are being created with the proper hardware addresses in each location, but the temperature readings are not being stored right, which is just strange. So for now I'm continuing to eat up some extra memory while I use the uthash library.

I'm trying to determine the best relay setup to use to control the attic fan. I guess I just need to get over it and by a $15-20 120-240V relay and be done with it.

# To Date min's, max's and observations

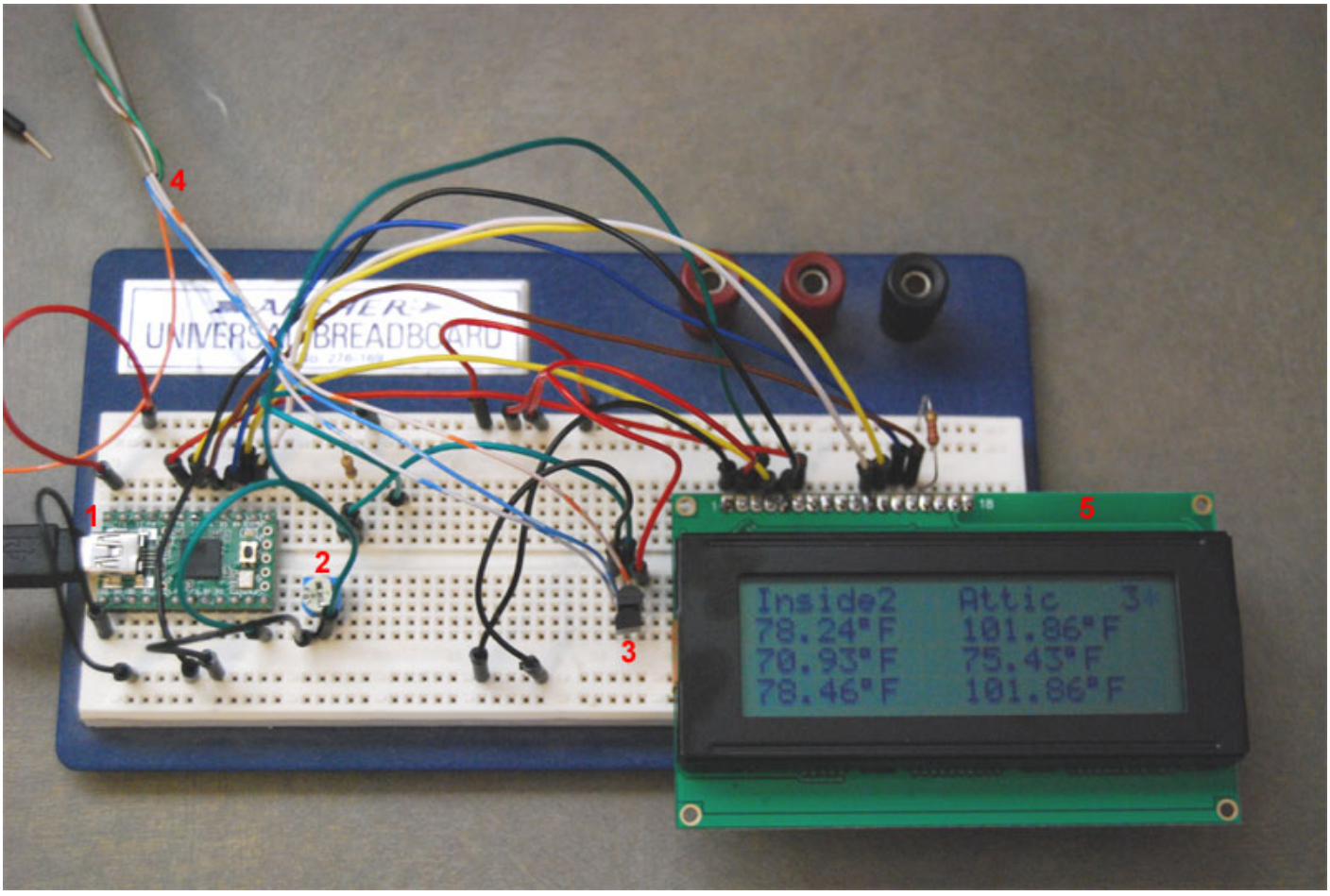
- 7/31/2011 14:11:16 - Max 131.67 @ Attic
- 8/3/2011 14:58:15 - Max: 135.16 @ Attic
- 1/16/2024 - Even though the outdoor temp was in the teens, the attic temp never dipped below freezing

- 1/21/2024 - Pending

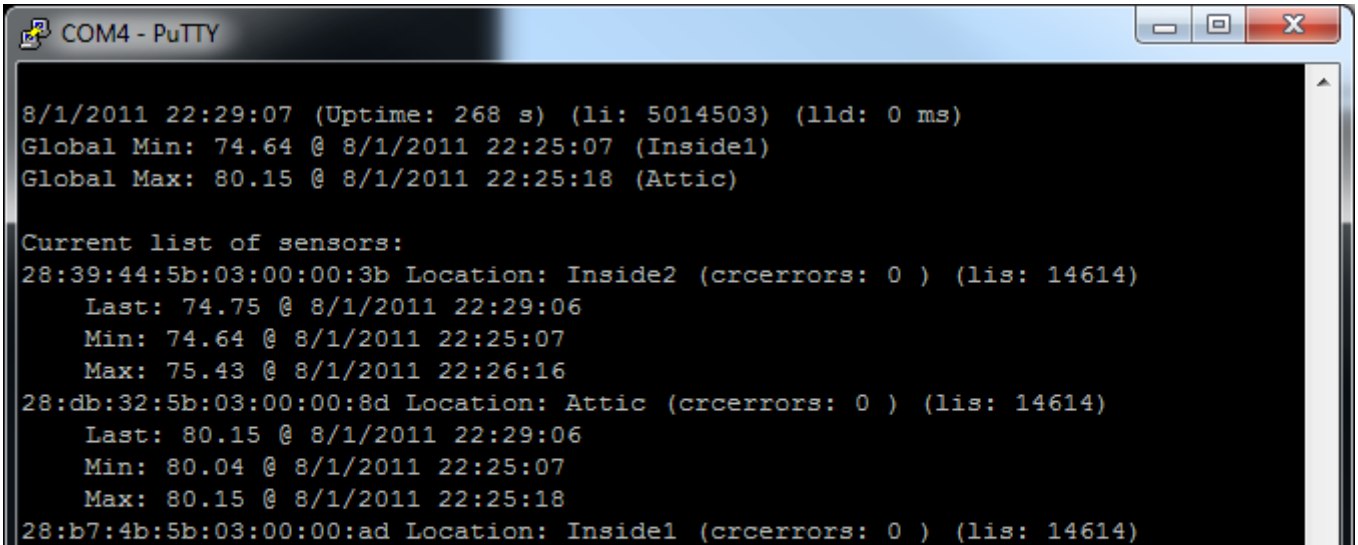
# Development Photos

Breadboarded project

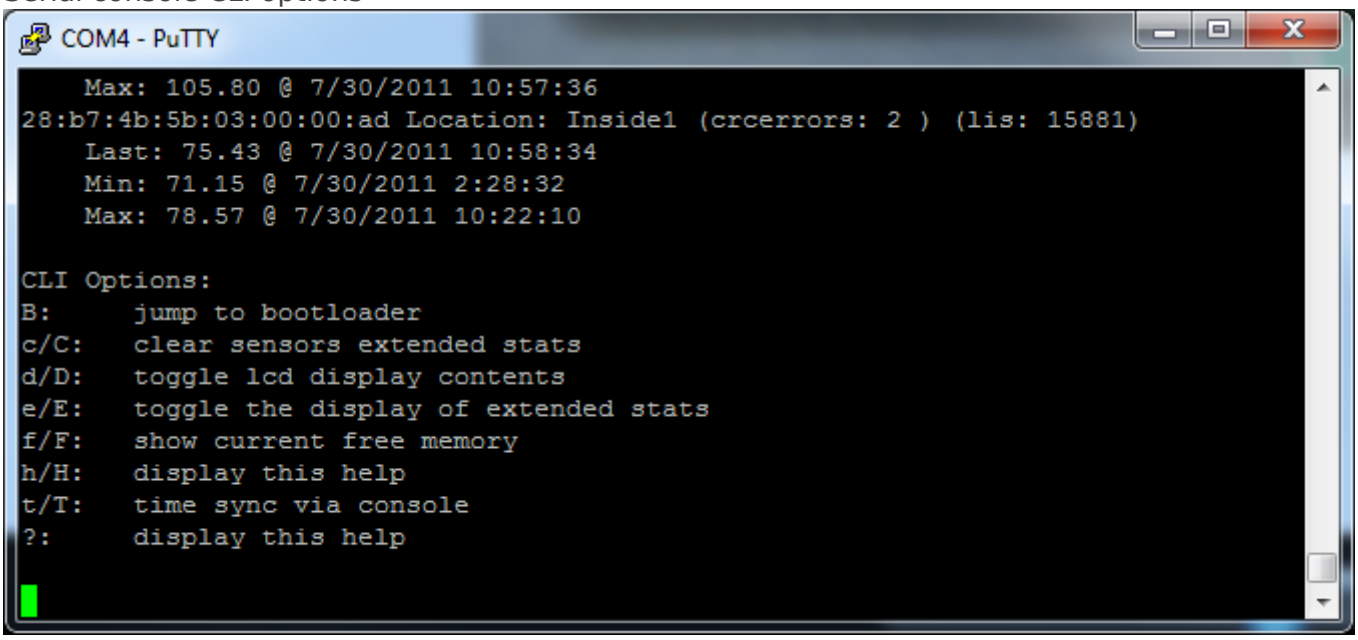
1. Teensy 2.0 Development Board
2. LCD Contrast Potentiometer (10k)
3. DS18B20 Programmable Resolution 1-Wire Digital Thermometer: two on board and one in attic
4. CAT3 extending 1-Wire bus to sensor in attic
5. HD44780 compatible LCD display

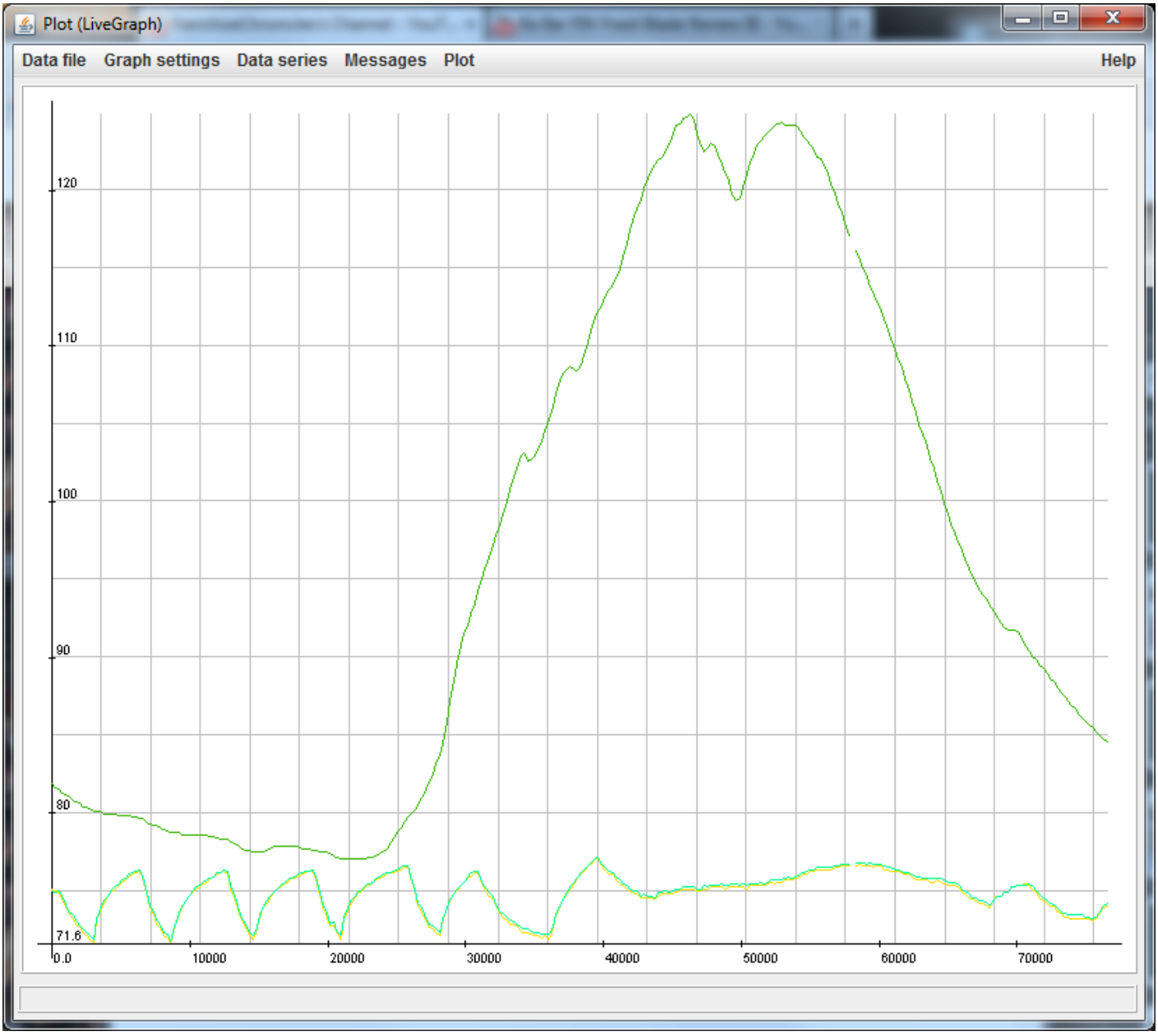



Serial console extended stats



Serial console CLI options



Graphed data from 8 Aug 2011 (dark green is attic temperature, others are inside temperature)

The CLI was updated to export data using a CSV format (millis,now(),each sensors last conversion...). A Python script was written to read the text from the serial interface and write it to a file. LiveGraph was used to read the file in real time and display the graphed data. The graph above begins at about 11:30 PM 7 Aug 2011 and ends about 11:30 PM 8 Aug 2011. One sample is output to the CLI every second. After roughly a 24 hour period the CSV file was 2.69MB in size.

# Version

The current version is 0.5c.

# Features

As of 0.5c (1 Aug 2011):

- Global min/max with sensor name and timestamps

As of 0.5b (30 July 2011):

- Dynamic sensor discovery on device power on
- Per sensor last/min/max temperatures with timestamps
- Non-blocking temperature conversions
- Sensor data display on LCD (currently limited to the first two sensors)
- Sensor data display on serial console (basic and extended)
- Serial console control
- Time update via serial console

# To Do

- Air Conditioner Filter Change Reminder
- Attic fan relay control
- EEPROM storage of configuration information
- Ethernet TCP/IP Interface using WizNet WIZ812MJ module
- Historical statistics other than just current/min/max
  - avg/min/max per day
  - long term five minute interval graphing (via external means, SNMP possible)
- Keypad for complete device control
- PCB design
- Final migration from breadboard to PCB

# Source Code

```
/*
Starting point:
 http://tushev.org/articles/electronics/42-how-it-works-ds18b20-and-arduino
 20150822 Forked from temp_05b_hash_realtime_metrof

 -= TODO =-

 A/C Filter Change Reminder

 -= EEPROM STORAGE MAP =-
 1k bytes EEPROM available
 Page size is 8 bytes
 128 Pages

 Reserve the first page for empty (due to slot 0 possiblity of being corrupted)

 Reserve pages 2-9 for configuration data

 TIMEZONE_OFFSET_HOURS - char
 DST - daylight savings time - byte
 NUM_STORED_SENSORS - byte
 AIRFILTER REPLACEMENT DATE
 AIRFILTER LIFESPAN

 Pages 10 and up are reserved for stored sensors

 STORED SENSORS:
 ADDR - byte[8] (1 page)
 NAME - char[10] (1 page plus 2 bytes)
 CALIBRATION_OFFSET - float (2 bytes)

 */

/*
 * Teensy 2.0 pinout details for this project
 * ==========================================
 * 0  -
```

```
 * 1  -
 * 2  -
 * 3  -
 * 4  -
 * 5  - * i2c SCL
 * 6  - * i2c SDA
 * 7  -
 * 8  -
 * 9  -
 * 10 - * OneWire bus
 * 11 - * LED_PIN Used for onboard signalling LED
 * 12 - * LCD Display
 * 13 - * LCD Display
 * 14 - * LCD Display
 * 15 - * LCD Display
 * 16 - * LCD Display
 * 17 - * LCD Display
 * 18 -
 * 19 -
 * 20 -
 * 21 - A0 - Connected to Adafruit GA1A12S202 Log-scale Analog Light Sensor (response from 3
to 55,000 lux)
 *
 * i2c Device Addresses
 * ====================
 * 0x29 - Adafruit TSL2591 High Dynamic Range Digital Light Sensor
 *
 */

// Uncomment the line below to enable DEBUG information. This will cause the compiled code to
increase.
//#define DEBUG 0

#define SKETCHVERSION 7

#define ACTIVITY_CHAR_INTERVAL 250
#define BLINK_INTERVAL 250
#define CONSOLE_UPDATE_INTERVAL 1000
#define DEFAULT_MIN_TEMP 999
#define DEFAULT_MAX_TEMP -99
```

```
#define DEFAULT_TIME_ZONE_OFFSET -5  // CST during DST
#define DS18B20_ID 0x28
#define DS18B20_CONVERSION_WAIT_TIME 750
#define HELP_PAUSE_METRO_INTERVAL 10000
#define LCD_CLEAR_INTERVAL 10000
#define LCD_UPDATE_INTERVAL 1000
#define LCD_STRING_BUFFER_LENGTH 21
#define LED_PIN 11 // physical pin 11

#define ANALOG_LUX_SENSOR_PIN A0 // physical pin 21
#define LIGHTANALOGSENSOR 250
#define LIGHTDIGITALSENSOR 1000

#include <avr/pgmspace.h>
#include <MemoryFree.h>
#include <LiquidCrystal.h>
#include <Metro.h>
#include <OneWire.h>
#include <String.h>
#include "C:\ArduinoProjects\lib\Streaming.h"
#include <Time.h>
#include "C:\ArduinoProjects\lib\uthash-master\uthash.h"

// 20150822 MSHARP - Adding TSL2591 light sensor support
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_TSL2591.h"

enum console_mode_t {
  standard,
  extended,
  streaming
};

console_mode_t consoleMode = standard;

// ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
// Using PROGMEM ROM to store strings
// Use printPROGMEMString() to print these strings to the serial port
// ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== =====
```

```
const char version_line_1[] PROGMEM = "Sketch version: ";

const char message_settingup [] PROGMEM = "Setting up...";

const char message_timesync_waiting [] PROGMEM = "Waiting for time data in @time_t format...";
const char message_timesync_updated [] PROGMEM = "Sync message received and time updated.";
const char message_timesync_invalid [] PROGMEM = "Invalid sync message received.";

const char message_bootloader_jump_1 [] PROGMEM = "Jumping to bootloader in ";
const char message_bootloader_jump_2 [] PROGMEM = "Make sure you close your serial
console!!!";
const char message_bootloader_jump_jumping [] PROGMEM = "Jumping!";

const char message_pressanykeytoabort [] PROGMEM = "PRESS ANY KEY TO ABORT!!!";
const char message_aborted [] PROGMEM = "Aborted!";

// ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== =====

#define CLI_STRING_BUFFER_LENGTH 50
#define NUM_CLI_LINES 13

const char cli_line_1[] PROGMEM  = "CLI Options:";
const char cli_line_2[] PROGMEM  = "B:     jump to bootloader";
const char cli_line_3[] PROGMEM  = "c:     cycle console mode";
const char cli_line_4[] PROGMEM  = "C:     clear sensors stats";
const char cli_line_5[] PROGMEM  = "d/D:   toggle lcd display contents";
const char cli_line_6[] PROGMEM  = "f/F:   show current free memory";
const char cli_line_7[] PROGMEM  = "G:     reset global sensor stats";
const char cli_line_8[] PROGMEM  = "h:     display this help";
const char cli_line_9[] PROGMEM  = "l:     query analog light sensor and display result";
const char cli_line_10[] PROGMEM = "L:     query digital light sensor and display result";
const char cli_line_11[] PROGMEM = "s/S:   set console mode to streaming";
const char cli_line_12[] PROGMEM = "T:     time sync via console";
const char cli_line_13[] PROGMEM = "v/V:   show version";

const char* const cli_string_table[] PROGMEM =
{
  cli_line_1,
  cli_line_2,
```

```
  cli_line_3,
  cli_line_4,
  cli_line_5,
  cli_line_6,
  cli_line_7,
  cli_line_8,
  cli_line_9,
  cli_line_10,
  cli_line_11,
  cli_line_12,
  cli_line_13
};

// ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== =====

// These are sensors that I know I have. Eventually we'll store this in EEPROM programatically
byte tempSensorAttic[8] = {
  0x28, 0xdb, 0x32, 0x5b, 0x03, 0x00, 0x00, 0x8d
};
byte tempSensorInside1[8] = {
  0x28, 0xb7, 0x4b, 0x5b, 0x03, 0x00, 0x00, 0xad
};
byte tempSensorInside2[8] = {
  0x28, 0x39, 0x44, 0x5b, 0x03, 0x00, 0x00, 0x3b
};

boolean consolePaused = false;
boolean lcdDisplayAddresses = false;
boolean showExtendedStats = false;

unsigned int activityCharIndex = 0;
unsigned int ledStatus = 0;

unsigned long loopIteration = 0;
unsigned long loopStartMillis;
unsigned long lastLoopDuration = 0;

// LCD Display using 6 pins (12-17)
LiquidCrystal lcd(16, 17, 12, 13, 14, 15);
```

```
// Adafruit TSL2591 High Dynamic Range Digital Light Sensor
boolean tslFound = false;
Adafruit_TSL2591 tsl = Adafruit_TSL2591(2591);

// Global variables for templerature sensors
const int luxAnalogNumReadings = 10;
float luxAnalogReadings[luxAnalogNumReadings];
float luxAnalogRaw = 0.0;
int luxAnalogRawIndex = 0;
float luxAnalogRawTotal = 0.0;
float luxAnalogRawAverage = 0.0;
float luxAnalogLog = 0.0;
float luxAnalogLogSmoothed = 0.0;

uint32_t luxDigitalFullLuminosity;
uint16_t luxDigitalIR, luxDigitalFull, luxDigitalCalculated;

// Manually update this number when you add a Metro below
#define NUM_METROS 7

Metro activityCharMetro(ACTIVITY_CHAR_INTERVAL, false);
Metro consoleUpdateMetro(CONSOLE_UPDATE_INTERVAL);
Metro helpPauseMetro(HELP_PAUSE_METRO_INTERVAL, false);
Metro ledMetro(BLINK_INTERVAL, false);
Metro lcdUpdateMetro(LCD_UPDATE_INTERVAL);
Metro lcdClearMetro(LCD_CLEAR_INTERVAL);  // counter to clear the lcd every 10 seconds for
housekeeping
Metro owBusSearchMetro(600000, false);  //  counter to search the bus every 10 minutes for new
devices
Metro lightAnalogSensor(LIGHTANALOGSENSOR, false);
Metro lightDigitalSensor(LIGHTDIGITALSENSOR, false);

// OneWire bus on pin 10
OneWire ds(10);

float globalMin = DEFAULT_MIN_TEMP;
char * globalMinName = NULL;
time_t globalMinTimeStamp = (time_t) 0;
float globalMax = DEFAULT_MAX_TEMP;
char * globalMaxName = NULL;
```

```
time_t globalMaxTimeStamp = (time_t) 0;

struct DS18B20 {
  byte addr[8];              /* key */
  char name[10];
  boolean active;
  boolean converting;  /* true if a conversion has been requested */
  unsigned int crcerrors;
  unsigned long startConversionLI;  // loop iteration of the start conversion
  unsigned long liLastConversion;    // number of loop iterations for the last conversion
  Metro conversionTimer;
  float lastTemp;
  time_t lastTimeStamp;
  float minTemp;
  time_t minTimeStamp;
  float maxTemp;
  time_t maxTimeStamp;
  UT_hash_handle hh;         /* makes this structure hashable */
};

struct DS18B20 *tempSensors = NULL;

boolean compareByteArray(byte a1[], byte a2[]) {
  int arraySize = sizeof(a1) / sizeof(byte);
  if (sizeof(a1) != sizeof(a2)) {
    return false;
  }
  for (int i = 0; i < arraySize; i++) {
    if (a1[i] != a2[i]) {
      return false;
    }
  }
  return true;
}

void add_sensor(byte addr[]) {
  struct DS18B20 *s;

  s = (DS18B20 *) malloc(sizeof(struct DS18B20));
  for (int i = 0; i < 8; i++) {
```

```
    s->addr[i] = addr[i];
  }
  s->active = false;
  s->converting = false;
  s->crcerrors = 0;
  s->conversionTimer = Metro(DS18B20_CONVERSION_WAIT_TIME, false);
  s->liLastConversion = 0;
  s->lastTemp = 0.0;
  s->lastTimeStamp = now();
  s->minTemp = DEFAULT_MIN_TEMP;
  s->minTimeStamp = s->lastTimeStamp;
  s->maxTemp = DEFAULT_MAX_TEMP;
  s->maxTimeStamp = s->lastTimeStamp;
  if (compareByteArray(s->addr, tempSensorAttic)) {
    strcpy(s->name, "Attic");
  }
  else if (compareByteArray(s->addr, tempSensorInside1)) {
    strcpy(s->name, "Inside1");
  }
  else if (compareByteArray(s->addr, tempSensorInside2)) {
    strcpy(s->name, "Inside2");
  }
  else {
    strcpy(s->name, "Unknown");
  }
  HASH_ADD(hh, tempSensors, addr, sizeof(byte) * 8, s);
}

struct DS18B20 *find_sensor(byte addr[]) {
  struct DS18B20 *s;

#ifdef DEBUG
  Serial.print("find_sensor(");
  Serial.print(OneWireaddrtostring(addr, false));
  Serial.println(")");
#endif

  for (s = tempSensors; s != NULL; s = (DS18B20 *) s->hh.next) {
#ifdef DEBUG
    Serial.println("Comparing:");
```

```
    Serial.print(OneWireaddrtostring(s->addr, false));
    Serial.print(" to ");
    Serial.println(OneWireaddrtostring(addr, false));
#endif

    if (compareByteArray(addr, s->addr)) {
#ifdef DEBUG
      Serial.println("  match found... sensor already detected");
#endif
      return s;
    }
  }

#ifdef DEBUG
  Serial.println("no match found... returning NULL");
#endif
  return NULL;
}

void update_console() {
  struct DS18B20 *s;

  // Do a digital lux reading before we attempt displaying anything... this will make the
display update more pleasing
  doLuxReadingDigital();

  if ((consoleMode == standard) || (consoleMode == extended)) {
    // Now lets start putting stuff on the console
    serialPrintDateTime(now());
    Serial << " (Uptime: " << millis() / 1000 << " s) (li: " << loopIteration << ") (lld: " <<
lastLoopDuration << " ms)" << endl;
    Serial << "Global Min: " << globalMin << " @ ";
    serialPrintDateTime(globalMinTimeStamp);
    Serial << " (" << globalMinName << ")" << endl;
    Serial << "Global Max: " << globalMax << " @ ";
    serialPrintDateTime(globalMaxTimeStamp);
    Serial << " (" << globalMaxName << ")" << endl << endl;
    Serial.println("Current list of sensors:");

    for (s = tempSensors; s != NULL; s = (DS18B20 *) s->hh.next) {
```

```
      Serial.print(OneWireaddrtostring(s->addr, false));
      if (s->active) {
        if (consoleMode == standard) {
          // show standard information
          Serial.print(" ");
          Serial.print(s->lastTemp);
          Serial.print("/");
          Serial.print(s->minTemp);
          Serial.print("/");
          Serial.print(s->maxTemp);
          Serial.print(" F, Location: ");
          Serial.println(s->name);
        }
        else {
          // show extended stats
          Serial << " Location: " << s->name << " (crcerrors: " << s->crcerrors << " ) (lis: "
<< s->liLastConversion << ")" << endl;
          Serial << "    Last: " << s->lastTemp << " @ ";
          serialPrintDateTime(s->lastTimeStamp);
          Serial << endl;
          Serial << "    Min: " << s->minTemp << " @ ";
          serialPrintDateTime(s->minTimeStamp);
          Serial << endl;
          Serial << "    Max: " << s->maxTemp << " @ ";
          serialPrintDateTime(s->maxTimeStamp);
          Serial << endl;
        }
      }
      else {
        Serial.println(" is pending first read.");
      }
    }
    // Display light sensor data now
    Serial.println();
    displayAnalogLightSensorData();
    displayDigitalLightSensorData();
  }
  else {
    // streaming
    //        Serial << "@" << endl;
```

```
//     serialPrintDateTime(now());
Serial << millis() << "," << now();
Serial.print(",");
/*     Serial << "," << millis() / 1000 << endl;
     Serial << globalMin << ",";
     serialPrintDateTime(globalMinTimeStamp);
     Serial << "," << globalMinName << endl;
     Serial << globalMax << ",";
     serialPrintDateTime(globalMaxTimeStamp);
     Serial << "," << globalMaxName << endl;
     */
for (s = tempSensors; s != NULL; s = (DS18B20 *) s->hh.next) {
  //      Serial.print(OneWireaddrtostring(s->addr, false));
  //      Serial.print(",");
  Serial.print(s->lastTemp);
  if (s->hh.next != NULL) {
    Serial.print(",");
  }
  /*      serialPrintDateTime(s->lastTimeStamp);
   Serial.print(",");
   Serial.print(s->minTemp);
   Serial.print(",");
   serialPrintDateTime(s->minTimeStamp);
   Serial.print(",");
   Serial.print(s->maxTemp);
   Serial.print(",");
   serialPrintDateTime(s->maxTimeStamp);
   Serial.print(",");
   Serial.println(s->name); */
}

Serial.print(",LA,");
Serial.print(luxAnalogReadings[luxAnalogRawIndex]);
Serial.print(",");
Serial.print(luxAnalogRawAverage);
Serial.print(",");
Serial.print(pow(10, luxAnalogLog));
Serial.print(",");
Serial.print(pow(10, luxAnalogLogSmoothed));
```

```
    if (tslFound) {
      Serial.print(",LD,");
      Serial.print("IR,"); Serial.print(luxDigitalIR);  Serial.print(",");
      Serial.print("F,"); Serial.print(luxDigitalFull); Serial.print(",");
      Serial.print("V,"); Serial.print(luxDigitalFull - luxDigitalIR); Serial.print(",");
      Serial.print("L,"); Serial.print(luxDigitalCalculated);
    }
  }
  Serial.println();
}

boolean update_sensor(byte addr[], float temp) {
  struct DS18B20 *s;

  s = find_sensor(addr);
  if (s != NULL) {
    if (!s->active) {
      s->active = true;
    }

    s->lastTemp = temp;
    s->lastTimeStamp = now();

    if (temp < s->minTemp) {
      s->minTemp = temp;
      s->minTimeStamp = now();
    }

    if (temp < globalMin) {
      globalMin = temp;
      globalMinName = s->name;
      globalMinTimeStamp = now();
    }

    if (temp > s->maxTemp) {
      s->maxTemp = temp;
      s->maxTimeStamp = now();
    }

    if (temp > globalMax) {
```

```
      globalMax = temp;
      globalMaxName = s->name;
      globalMaxTimeStamp = now();
    }

    return true;
  }
  else {
#ifdef DEBUG
    Serial.print("update_sensor() failed for addr ");
    Serial.print(OneWireaddrtostring(addr, false));
    Serial.print(" ");
    Serial.print(temp);
    Serial.println(" F");
#endif
    return false;
  }
}

int count_sensors() {
  struct DS18B20 *s;
  int count = 0;
  for (s = tempSensors; s != NULL; s = (DS18B20 *) s->hh.next, count++) {
  }
  return count;
}

void clear_sensor_stats() {
  struct DS18B20 *s;

  for (s = tempSensors; s != NULL; s = (DS18B20 *) s->hh.next) {
    s->minTemp = DEFAULT_MIN_TEMP;
    s->minTimeStamp = (time_t) 0;
    s->maxTemp = DEFAULT_MAX_TEMP;
    s->maxTimeStamp = (time_t) 0;
  }
}

void clear_global_sensor_stats() {
  globalMin = DEFAULT_MIN_TEMP;
```

```
  globalMinName = NULL;
  globalMinTimeStamp = (time_t) 0;
  globalMax = DEFAULT_MAX_TEMP;
  globalMaxName = NULL;
  globalMaxTimeStamp = (time_t) 0;
}

void doLuxReadingAnalog() {
  float rawRange = 1024; // 3.3v
  float logRange = 5.0;  // 3.3 = 10^5 lux

  luxAnalogRawTotal = luxAnalogRawTotal - luxAnalogReadings[luxAnalogRawIndex];
  luxAnalogReadings[luxAnalogRawIndex] = analogRead(ANALOG_LUX_SENSOR_PIN);
  luxAnalogRawTotal = luxAnalogRawTotal + luxAnalogReadings[luxAnalogRawIndex];
  luxAnalogRawAverage = luxAnalogRawTotal / luxAnalogNumReadings;
  luxAnalogLog = luxAnalogRawAverage * logRange / rawRange;
  luxAnalogLogSmoothed = luxAnalogRawAverage * logRange / rawRange;

  luxAnalogRawIndex = luxAnalogRawIndex + 1;
  if (luxAnalogRawIndex >= luxAnalogNumReadings){
    luxAnalogRawIndex = 0;
  }

}

void displayAnalogLightSensorData() {
  Serial.print("Light [analog] : ");
  Serial.print("Raw: ");
  Serial.print(luxAnalogReadings[luxAnalogRawIndex]);
  Serial.print("  Smoothed: ");
  Serial.print(luxAnalogRawAverage);
  Serial.print("  Log: ");
  Serial.print(pow(10, luxAnalogLog));
  Serial.print("  Smoothed: ");
  Serial.println(pow(10, luxAnalogLogSmoothed));
}

// Updates global variables to be used in displayDigitalLightSensorDataData()
void doLuxReadingDigital() {
  if (!tslFound) {
```

```
    Serial.println("No TSL2591 device has been found.");
    return;
  }

  // You can change the gain on the fly, to adapt to brighter/dimmer light situations
  //tsl.setGain(TSL2591_GAIN_LOW);    // 1x gain (bright light)
  tsl.setGain(TSL2591_GAIN_MED);      // 25x gain
  //tsl.setGain(TSL2591_GAIN_HIGH);   // 428x gain

  // 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(TSL2591_INTEGRATIONTIME_100MS);  // shortest integration time (bright light)
  //tsl.setTiming(TSL2591_INTEGRATIONTIME_200MS);
  //tsl.setTiming(TSL2591_INTEGRATIONTIME_300MS);
  //tsl.setTiming(TSL2591_INTEGRATIONTIME_400MS);
  //tsl.setTiming(TSL2591_INTEGRATIONTIME_500MS);
  //tsl.setTiming(TSL2591_INTEGRATIONTIME_600MS);  // longest integration time (dim light)

  luxDigitalFullLuminosity = tsl.getFullLuminosity();
  luxDigitalIR = luxDigitalFullLuminosity >> 16;
  luxDigitalFull = luxDigitalFullLuminosity & 0xFFFF;
  luxDigitalCalculated = tsl.calculateLux(luxDigitalFull, luxDigitalIR);
}

void displayDigitalLightSensorData() {
  if (!tslFound) {
    Serial.println("No TSL2591 device has been found.");
    return;
  }

  Serial.print("Light [digital]: ");
  Serial.print("IR: "); Serial.print(luxDigitalIR);  Serial.print("  ");
  Serial.print("Full: "); Serial.print(luxDigitalFull); Serial.print("  ");
  Serial.print("Visible: "); Serial.print(luxDigitalFull - luxDigitalIR); Serial.print("  ");
  Serial.print("Lux: "); Serial.print(luxDigitalCalculated);
  Serial.println();
}

boolean findDS18B20Devices(OneWire & ow) {
  byte addr[8];
```

```
  unsigned int deviceCount = 0;

#ifdef DEBUG
  Serial.println("Searching bus...");
#endif

  //find a device
  while (ow.search(addr)) {
    if (OneWire::crc8( addr, 7) != addr[7]) {
      Serial.println("Bad crc!!!");
      continue;
    }

    if (addr[0] != DS18B20_ID) {
#ifdef DEBUG
      Serial.print("Unknown device: ");
#endif
      Serial.println(OneWireaddrtostring(addr, false));
      continue;
    }

#ifdef DEBUG
    Serial.print("Found a device:");
    Serial.println(OneWireaddrtostring(addr, false));
#endif

    deviceCount++;
    if (find_sensor(addr) == NULL) {
#ifdef DEBUG
      Serial.print("Adding new sensor to list: ");
#endif
      add_sensor(addr);
    }
#ifdef DEBUG
    else {
      Serial.print("We already know about this sensor: ");
    }
#endif
    Serial.println(OneWireaddrtostring(addr, false));
    Serial.println();
```

```
  }

#ifdef DEBUG
  Serial.println("No more devices found... resetting search.");
  Serial.println();
#endif
  ow.reset_search();
}

boolean requestTemperatureConversion(OneWire ow, DS18B20 *sensor) {
  ow.reset();
  ow.select(sensor->addr);
  ow.write(0x44, 1);

  return true;
}

float retrieveTemperature(OneWire ow, DS18B20 *sensor) {
  byte data[12];
  float temp;

  ow.reset();
  ow.select(sensor->addr);
  ow.write(0xBE);
  for (int i = 0; i < 9; i++) {
    data[i] = ow.read();
  }

  temp = ( (data[1] << 8) + data[0] ) * 0.0625; // tempc
  temp = (temp * 1.8) + 32;  // tempf

  if (OneWire::crc8( data, 8) != data[8]) {
    temp = -9999;
  }

  return temp;
}

void toggleLED() {
  switch (ledStatus) {
```

```
    case 1:
      digitalWrite(LED_PIN, LOW);
      ledStatus = 0;
      break;
    default:
      digitalWrite(LED_PIN, HIGH);
      ledStatus = 1;
      break;
  }
}

void showActivityChar() {
  lcd.setCursor(18, 0);
  switch (activityCharIndex) {
    case 1:
      activityCharIndex--;
      lcd.print(count_sensors());
      lcd.print(" ");
      break;
    default:
      activityCharIndex++;
      lcd.print(count_sensors());
      lcd.print("*");
      break;
  }
}

String OneWireaddrtostring(byte addr[], boolean lcd) {
  String toReturn;

  for ( int i = 0; i < 8; i++) {
    if (addr[i] < 16) {
      toReturn += "0";
    }
    toReturn += String(addr[i], HEX);
    if (i < 7) {
      if (!lcd) {
        // don't print semicolons on the lcd
        // we don't have enough room
        toReturn += ":";
```

```
      }
    }
  }

  return toReturn;
}

void update_lcd() {
  struct DS18B20 *s;
  unsigned int count;
  unsigned int maxCount = 2;

  s = tempSensors;
  if (lcdDisplayAddresses) {
    maxCount = 3;
  }

  if (lcdClearMetro.check() == 1) {
    lcd.clear();
    lcdClearMetro.reset();
  }

  if (lcdDisplayAddresses) {
    lcd.setCursor(0, 0);
    lcd.print("Uptime: ");
    lcd.print(millis() / 1000);
    lcd.print("s ");
  }

  for (count = 0; count < maxCount; count++, s = (DS18B20 *) s->hh.next) {
    if (s == NULL) {
      break;
    }

    if (!lcdDisplayAddresses) {
      if (!s->active) {
        continue;
      }
      lcd.setCursor(count * 10, 0);
      lcd.print(s->name);
```

```
      lcd.setCursor(count * 10, 1);
      lcd.print(s->lastTemp);
      lcd.print((char)223);
      lcd.print("F");
      lcd.setCursor(count * 10, 2);
      lcd.print(s->minTemp);
      lcd.print((char)223);
      lcd.print("F");
      lcd.setCursor(count * 10, 3);
      lcd.print(s->maxTemp);
      lcd.print((char)223);
      lcd.print("F");
    }
    else {
      // display addresses instead of temps
      lcd.setCursor(0, count + 1);
      lcd.print("    ");
      lcd.print(OneWireaddrtostring(s->addr, true));
      lcd.setCursor(0, count + 1);
      lcd.print(s->name);
      lcd.print(":");
    }
  }
}

void serialPrintDateTime(time_t timeStamp) {
  time_t timeStampAdjusted = timeStamp + (DEFAULT_TIME_ZONE_OFFSET * 60 * 60);
  Serial << month(timeStampAdjusted) << "/" << day(timeStampAdjusted) << "/" <<
year(timeStampAdjusted) << " " << hour(timeStampAdjusted) << ":";
  if (minute(timeStampAdjusted) < 10) Serial << "0";
  Serial << minute(timeStampAdjusted) << ":";
  if (second(timeStampAdjusted) < 10) Serial << "0";
  Serial << second(timeStampAdjusted);
}

void processTimeSyncMessage() {
  int count = 0;
  char buf[11];
  boolean status = false;  // did we get good data
  printPROGMEMString(message_timesync_waiting); // "Waiting for time data in @time_t
```

```
format..."
  Serial.println();
  Serial.flush();
  while (count < 11) {
    if (Serial.available()) {  // receive all 11 bytes into "buf"
      buf[count++] = Serial.read();
    }
  }
  if (buf[0] == '@') {
    time_t pctime = 0;
    for (int i = 1; i < 11; i++) {
      char c = buf[i];
      if (c >= '0' && c <= '9') {
        pctime = (10 * pctime) + (c - '0') ; // convert digits to a number
      }
    }
    pctime += 10;
    setTime(pctime);   // Sync clock to the time received
    status = true;
  }
  if (status) {
    // "Sync message received and time updated."
    printPROGMEMString(message_timesync_updated);
    Serial.println();
  }
  else {
    // "Invalid sync message received."
    printPROGMEMString(message_timesync_invalid);
    Serial.println();
  }
}

void jumpToBootloader() {
  unsigned int counter = 10;
  printPROGMEMString(message_bootloader_jump_1); // "Jumping to bootloader in "
  Serial << counter << " seconds...";
  Serial.println();
  printPROGMEMString(message_bootloader_jump_2); // "Make sure you close your serial
console!!!"
  Serial.println(); Serial.println();
```

```
  printPROGMEMString(message_pressanykeytoabort); // "PRESS ANY KEY TO ABORT!!!"
  Serial.println(); Serial.println();
  Serial.flush();
  lcd.clear();
  while (counter > 0) {
    lcd.setCursor(0, 0);
    lcd.print(counter);
    Serial << counter << " ";
    if (Serial.available()) {
      Serial.flush();
      Serial.println(); Serial.println();
      printPROGMEMString(message_aborted); // "Aborted!"
      Serial.println(); Serial.println(); Serial.println();
      return;
    }
    delay(1000);
    counter--;
  }
  printPROGMEMString(message_bootloader_jump_jumping); // "Jumping!"
  Serial.println();
  cli();
  // disable watchdog, if enabled
  // disable all peripherals
  UDCON = 1;
  USBCON = (1 << FRZCLK); // disable USB
  UCSR1B = 0;
  delay(5);
#if defined(__AVR_AT90USB162__)                // Teensy 1.0
  EIMSK = 0; PCICR = 0; SPCR = 0; ACSR = 0; EECR = 0;
  TIMSK0 = 0; TIMSK1 = 0; UCSR1B = 0;
  DDRB = 0; DDRC = 0; DDRD = 0;
  PORTB = 0; PORTC = 0; PORTD = 0;
  asm volatile("jmp 0x3E00");
#elif defined(__AVR_ATmega32U4__)              // Teensy 2.0
  EIMSK = 0; PCICR = 0; SPCR = 0; ACSR = 0; EECR = 0; ADCSRA = 0;
  TIMSK0 = 0; TIMSK1 = 0; TIMSK3 = 0; TIMSK4 = 0; UCSR1B = 0; TWCR = 0;
  DDRB = 0; DDRC = 0; DDRD = 0; DDRE = 0; DDRF = 0; TWCR = 0;
  PORTB = 0; PORTC = 0; PORTD = 0; PORTE = 0; PORTF = 0;
  asm volatile("jmp 0x7E00");
#elif defined(__AVR_AT90USB646__)              // Teensy++ 1.0
```

```
  EIMSK = 0; PCICR = 0; SPCR = 0; ACSR = 0; EECR = 0; ADCSRA = 0;
  TIMSK0 = 0; TIMSK1 = 0; TIMSK2 = 0; TIMSK3 = 0; UCSR1B = 0; TWCR = 0;
  DDRA = 0; DDRB = 0; DDRC = 0; DDRD = 0; DDRE = 0; DDRF = 0;
  PORTA = 0; PORTB = 0; PORTC = 0; PORTD = 0; PORTE = 0; PORTF = 0;
  asm volatile("jmp 0xFC00");
#elif defined(__AVR_AT90USB1286__)             // Teensy++ 2.0
  EIMSK = 0; PCICR = 0; SPCR = 0; ACSR = 0; EECR = 0; ADCSRA = 0;
  TIMSK0 = 0; TIMSK1 = 0; TIMSK2 = 0; TIMSK3 = 0; UCSR1B = 0; TWCR = 0;
  DDRA = 0; DDRB = 0; DDRC = 0; DDRD = 0; DDRE = 0; DDRF = 0;
  PORTA = 0; PORTB = 0; PORTC = 0; PORTD = 0; PORTE = 0; PORTF = 0;
  asm volatile("jmp 0x1FC00");
#endif
}

void printCLIOptions() {
//  char buf[CLI_STRING_BUFFER_LENGTH];

  for (int i = 0; i < NUM_CLI_LINES; i++) {
//    strcpy_P(buf, (char*)pgm_read_word(&(cli_string_table[i])));
//    Serial.println(buf);
    printPROGMEMString((char*) pgm_read_word(&(cli_string_table[i])));
    Serial.println();
  }

  Serial.println();
}

// 20150822 MSHARP - Added function
void printPROGMEMString(const char* PMSTRING) {
  int i;
  int len = strlen_P(PMSTRING);
  char nextCharacter;
  for (i = 0; i < len; i++) {
    nextCharacter = pgm_read_byte_near(PMSTRING + i);
    Serial.print(nextCharacter);
  }

}

void setup() {
```

```
  // Initialize the display and tell the world we're starting to work
  lcd.begin(20, 4);
  lcd.setCursor(0, 0);
  lcd.print("Setting up...");

  pinMode(LED_PIN, OUTPUT);
  ledMetro.reset();
  Serial.begin(9600);

  // displaying activity char so we know we've started the initial 1-wire search
  showActivityChar();

  // Give human 10 seconds to connect via serial to watch for debug information
#ifdef DEBUG
  for (int i = 0; i < 10; i++) {
    Serial << i << " ";
    delay(1000);
  }
  Serial << endl;
#endif

  findDS18B20Devices(ds);

  // Setup Adafruit i2c TSL sensor
  if (tsl.begin()) {
    Serial.println("Found a TSL2591 sensor!!!");
    tslFound = true;
  }

  // Give human 10 seconds to view debug information from device scan
#ifdef DEBUG
  for (int i = 0; i < 10; i++) {
    Serial << i << " ";
    delay(1000);
  }
  Serial << endl;
#endif

  // Initialize analog lux reading smoothing array
  for (int thisReading = 0; thisReading < luxAnalogNumReadings; thisReading++) {
```

```
    luxAnalogReadings[thisReading] = 0.0;
  }
} // end setup

void loop() {
  loopIteration++;
  loopStartMillis = millis();

  float tmpTemp = 0;
  struct DS18B20 *s;

  if (activityCharMetro.check() == 1) {
    showActivityChar();
    activityCharMetro.reset();
  }

  // manage the sensors
  for (s = tempSensors; s != NULL; s = (DS18B20 *) s->hh.next) {
    if (s->converting) {
      if (s->conversionTimer.check() == 1) {
        tmpTemp = retrieveTemperature(ds, s);
        if (tmpTemp != -9999) {
          update_sensor(s->addr, tmpTemp);
        }
        else {
          s->crcerrors++;
        }
        s->liLastConversion = loopIteration - s->startConversionLI;
        s->converting = false;
        tmpTemp = 0;
      }
    }
    else {
      requestTemperatureConversion(ds, s);
      s->startConversionLI = loopIteration;
      s->conversionTimer.reset();
      s->converting = true;
    }

  } // for tempSensors
```

```
  // manage analog lux sensor
  if (lightAnalogSensor.check() == 1) {
    doLuxReadingAnalog();
    lightAnalogSensor.reset();
  }


  // process command line input
  if (Serial.available() > 0) {
    char c = Serial.read();
    switch (c) {
      case 'B':
        jumpToBootloader();
        break;
      case 'c':
        if (consoleMode == streaming) {
#ifdef DEBUG
          Serial.println("Setting console mode to standard.");
#endif
          consoleMode = standard;
        }
        else if (consoleMode == standard) {
#ifdef DEBUG
          Serial.println("Setting console mode to extended.");
#endif
          consoleMode = extended;
        }
        else if (consoleMode == extended) {
#ifdef DEBUG
          Serial.println("Setting console mode to streaming.");
#endif
          consoleMode = streaming;
        }
        break;
      case 'C':
        clear_sensor_stats();
        break;
      case 'd':
      case 'D':
```

```
        if (lcdDisplayAddresses) {
#ifdef DEBUG
          Serial.println("Switching LCD to display sensor values");
#endif
          lcdDisplayAddresses = false;
        }
        else {
#ifdef DEBUG
          Serial.println("Switching LCD to display sensor addresses");
#endif
          lcdDisplayAddresses = true;
        }
        Serial.println();
        Serial.flush();
        lcd.clear();
        break;
      case 'f':
      case 'F':
        Serial << "Free memory: " << freeMemory() << " bytes." << endl;
#ifdef DEBUG
        Serial << sizeof(DS18B20) * count_sensors() << " bytes for " << count_sensors() << "
DS18B20 sensors" << endl;
        Serial << sizeof(LiquidCrystal) << " bytes for LiquidCrystal object" << endl;
        Serial << sizeof(Metro) * NUM_METROS << " bytes for " << NUM_METROS << " Metro
objects" << endl;
        Serial << sizeof(OneWire) << " bytes for OneWire object" << endl;
#endif
        Serial << endl;
        break;
      case 'G':
        clear_global_sensor_stats();
        break;
      case 'l':
        displayAnalogLightSensorData();
        break;
      case 'L':
        displayDigitalLightSensorData();
        break;
      case 's':
      case 'S':
```

```
#ifdef DEBUG
        Serial.println("Setting console mode to streaming.");
#endif
        consoleMode = streaming;
        break;
      case 'T':
#ifdef DEBUG
        Serial.println("Processing time sync request...");
#endif
        processTimeSyncMessage();
#ifdef DEBUG
        Serial.println("Done!");
#endif
        Serial.println();
        break;
      case 'h':
      case 'H':
      case '?':
        printCLIOptions();
        // consoleUpdateMetro.autoreset(false); // 20150822 MSHARP commented out because
autoreset is now private method
        consoleUpdateMetro.reset();
        helpPauseMetro.reset();
        consolePaused = true;
        break;
      case 'v':
      case 'V':
        printPROGMEMString(version_line_1);
        Serial.print(" ");
        Serial.println(SKETCHVERSION);
        Serial.println();
        break;
      default:
        Serial.print("Key: 0x");
        Serial.println(c, HEX);
        printCLIOptions();
    } // testing c
  }

  if (helpPauseMetro.check() == 1) {
```

```
    consolePaused = false;
    // consoleUpdateMetro.autoreset(true); // 20150822 MSHARP commented out because autoreset
is now private method
    consoleUpdateMetro.reset();
  }

  if ((!consolePaused) && (consoleUpdateMetro.check() == 1)) {
    update_console();
  }

  if (lcdUpdateMetro.check() == 1) {
    update_lcd();
  }

  if (ledMetro.check() == 1) {
    toggleLED();
    ledMetro.reset();
  }

  lastLoopDuration = millis() - loopStartMillis;
} // end loop
```

Couldn't find my previous Python script so I had to crank out a new one quickly.

```
# Python 3.12 venv setup
mkdir -p ~/dev/python/python-teensyclimate-interface
cd ~/dev/python/python-teensyclimate-interface
python3.12 -m venv ven
source ./venv/bin/activate
pip install pyserial
```

```
# file:teensyclimate-log.py
# MSHARP 20240115
# 1. Open serial port
# 2. Write time sycn string to serial port
# 3. Start logging data to console
#    Intent is to SSH into remote machine connected to microcontroller
#    and log the session output to another computer, say via PuTTy
```

```
import serial
import time

# open serial port
teensy = serial.Serial('/dev/ttyACM0', 115200, timeout=3)

# send time sync command with time data
timeSyncString = "@"+str(time.time())[0:10]
print(str(f"Writing time sync string {timeSyncString} to serial port..."))
teensy.write(f'T'.encode())
teensy.write(f'{timeSyncString}'.encode())

print(f'Response:')

# loop to read responses from serial port
while True:
    try:
        line = teensy.readline()
        if (line != ""):
            print(line.rstrip().decode())
    except KeyboardInterrupt:
        print()
        print("CTRL-C received")
        break;

print("Shutting down...")
teensy.close()
```

#end

---

Revision #6
Created 12 August 2018 10:14:48 by bluecrow76
Updated 20 January 2025 16:50:39 by bluecrow76