第 1 章

付録

1.1　ソースコード

1.1.1　atmega328p-au(ニキシー管制御) 向けソース

Source Code 1.1　atmega328_firm.ino

```
#include <Wire.h>
#include <MsTimer2.h>
#include <SerialCommand.h>

volatile char num_signal_pattern[11][2] = {
  {0b00000000, 0b10000000}, // 0
  {0b00000100, 0b00000000}, // 1
  {0b00001000, 0b00000000}, // 2
  {0b00000000, 0b00000001}, // 3
  {0b00000000, 0b00000010}, // 4
  {0b00000000, 0b00000100}, // 5
  {0b00000000, 0b00001000}, // 6
  {0b00000000, 0b00010000}, // 7
  {0b00000000, 0b00100000}, // 8
  {0b00000000, 0b01000000}, // 9
  {0b00000000, 0b00000000}, // _
};

volatile char dot_signal_pattern[4][2] = {
  {0b00000000, 0b00000000}, // 00
  {0b00000001, 0b00000000}, // 01
  {0b00000010, 0b00000000}, // 10
  {0b00000011, 0b00000000}, // 11
};



int cycleNum = 0;
volatile char display_pattern[8][2] = {
  {0b00000000, 0b10000000}, // nix0
  {0b00000000, 0b01000000}, // nix1
  {0b00000001, 0b00000000}, // nix2
  {0b00000000, 0b00000100}, // nix3
  {0b00000000, 0b00010000}, // nix4
  {0b00000001, 0b00000000}, // nix5
  {0b00000000, 0b10000000}, // nix6
  {0b00000000, 0b10000000}, // nix7
};

volatile bool anode_signal_pattern[8][8] = {
  {1,1,1,1,1,1,1,0},
  {1,1,1,1,1,1,0,1},
  {1,1,1,1,1,0,1,1},
  {1,1,1,1,0,1,1,1},
  {1,1,1,0,1,1,1,1},
  {1,1,0,1,1,1,1,1},
  {1,0,1,1,1,1,1,1},
  {0,1,1,1,1,1,1,1},
};

volatile char cycle = 0;
volatile char before_cycle = 0;


void timer_interrupt(){
  // anode select counter
  cycle++;
  if(cycle > 7){
    cycle = 0;
  }

  digitalWrite(before_cycle+2,1);

  shiftOut(10,11,MSBFIRST, display_pattern[cycle][0]);
  shiftOut(10,11,MSBFIRST, display_pattern[cycle][1]);

  digitalWrite(12,LOW);
  digitalWrite(12,HIGH);
  digitalWrite(12,LOW);

  // force
  /*
  digitalWrite(2,anode_signal_pattern[cycle][0]);
  digitalWrite(3,anode_signal_pattern[cycle][1]);
  digitalWrite(4,anode_signal_pattern[cycle][2]);
  digitalWrite(5,anode_signal_pattern[cycle][3]);
  digitalWrite(6,anode_signal_pattern[cycle][4]);
  digitalWrite(7,anode_signal_pattern[cycle][5]);
  digitalWrite(8,anode_signal_pattern[cycle][6]);
  digitalWrite(9,anode_signal_pattern[cycle][7]);
  */

  digitalWrite(cycle+2,0);

  before_cycle = cycle;
}

// serial command control object
SerialCommand SCmd;

void setup(){
  Wire.begin();

  Serial.begin(9600);

  // serial command
  uint8_t steps = 0;
  uint8_t addr = 0x2F;

  Wire.beginTransmission(addr);
  Wire.write(steps);
  Wire.endTransmission();

  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);

  // mode
  pinMode(13,OUTPUT);
  digitalWrite(13, HIGH);

  SCmd.addCommand("num", set_num);
  SCmd.addCommand("dot", set_dot);
  SCmd.addCommand("dcdc_on", dcdc_on);
  SCmd.addCommand("dcdc_off", dcdc_off);
  SCmd.addDefaultHandler(error);


  MsTimer2::set(0.1,timer_interrupt);
  MsTimer2::start();
}

int translate_num(char input_char){
  switch(input_char){
    case '0':
      return 0;
      break;

    case '1':
      return 1;
      break;

    case '2':
      return 2;
      break;

    case '3':
      return 3;
      break;

    case '4':
      return 4;
      break;

    case '5':
      return 5;
      break;

    case '6':
      return 6;
      break;

    case '7':
      return 7;
      break;

    case '8':
      return 8;
      break;

    case '9':
      return 9;
      break;

    case 'n':
      return 10;
      break;
}
}



void set_num(){
  char dot_save[8] = {
    0, // nix0
    0, // nix1
    0, // nix2
    0, // nix3
    0, // nix4
    0, // nix5
    0, // nix6
    0, // nix7
```

```
  };

  char* arg = SCmd.next();

  //save dot
  for(char n = 0; n < 8; n++){
    dot_save[n] = (display_pattern[n][0] & 0b00000011);
  }

  memcpy(display_pattern[0], (void*)num_signal_pattern[translate_num(arg[0])], 2);
  memcpy(display_pattern[1], (void*)num_signal_pattern[translate_num(arg[1])], 2);
  memcpy(display_pattern[2], (void*)num_signal_pattern[translate_num(arg[2])], 2);
  memcpy(display_pattern[3], (void*)num_signal_pattern[translate_num(arg[3])], 2);
  memcpy(display_pattern[4], (void*)num_signal_pattern[translate_num(arg[4])], 2);
  memcpy(display_pattern[5], (void*)num_signal_pattern[translate_num(arg[5])], 2);
  memcpy(display_pattern[6], (void*)num_signal_pattern[translate_num(arg[6])], 2);
  memcpy(display_pattern[7], (void*)num_signal_pattern[translate_num(arg[7])], 2);

  //recover dot
  for(char n = 0; n < 8; n++){
    display_pattern[n][0] = (display_pattern[n][0] | dot_save[n]);
  }
}

void set_dot(){
  char* arg = SCmd.next();

  display_pattern[0][0] = ((display_pattern[0][0] & 0b11111100) | dot_signal_pattern[
        translate_num(arg[0])][0]);
  display_pattern[1][0] = ((display_pattern[1][0] & 0b11111100) | dot_signal_pattern[
        translate_num(arg[1])][0]);
  display_pattern[2][0] = ((display_pattern[2][0] & 0b11111100) | dot_signal_pattern[
        translate_num(arg[2])][0]);
  display_pattern[3][0] = ((display_pattern[3][0] & 0b11111100) | dot_signal_pattern[
        translate_num(arg[3])][0]);
  display_pattern[4][0] = ((display_pattern[4][0] & 0b11111100) | dot_signal_pattern[
        translate_num(arg[4])][0]);
  display_pattern[5][0] = ((display_pattern[5][0] & 0b11111100) | dot_signal_pattern[
        translate_num(arg[5])][0]);
  display_pattern[6][0] = ((display_pattern[6][0] & 0b11111100) | dot_signal_pattern[
        translate_num(arg[6])][0]);
  display_pattern[7][0] = ((display_pattern[7][0] & 0b11111100) | dot_signal_pattern[
        translate_num(arg[7])][0]);
}

void dcdc_on(){
  digitalWrite(13, HIGH);
}

void dcdc_off(){
  digitalWrite(13, LOW);
}

```

```
void error(){
  Serial.println("ubnrecongized␣command");
}

void loop(){
  SCmd.readSerial();
}
```

Source Code 1.2 SerialCommand.h

```
/*******************************************************************************
SerialCommand - An Arduino library to tokenize and parse commands received over
a serial port.
Copyright (C) 2011-2013 Steven Cogswell <steven.cogswell@gmail.com>
http://awtfy.com

Version 20131021A.

Version History:
May 11 2011 - Initial version
May 13 2011 - Prevent overwriting bounds of SerialCommandCallback[] array in addCommand()
                        defaultHandler() for non-matching commands
Mar 2012 - Some const char * changes to make compiler happier about deprecated warnings.
            Arduino 1.0 compatibility (Arduino.h header)
Oct 2013 - SerialCommand object can be created using a SoftwareSerial object, for SoftwareSerial
            support. Requires #include <SoftwareSerial.h> in your sketch even if you don't use
            a SoftwareSerial port in the project. sigh. See Example Sketch for usage.
Oct 2013 - Conditional compilation for the SoftwareSerial support, in case you really, really
            hate it and want it removed.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
*******************************************************************************/
#ifndef SerialCommand_h
#define SerialCommand_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

// If you want to use SerialCommand with the hardware serial port only, and want to disable
// SoftwareSerial support, and thus don't have to use "#include <SoftwareSerial.h>" in your
// sketches, then uncomment this define for SERIALCOMMAND_HARDWAREONLY, and comment out the
// corresponding #undef line.
//
// You don't have to use SoftwareSerial features if this is not defined, you can still only use
// the Hardware serial port, just that this way lets you get out of having to include
// the SoftwareSerial.h header.
//#define SERIALCOMMAND_HARDWAREONLY 1
#undef SERIALCOMMAND_HARDWAREONLY

#ifdef SERIALCOMMAND_HARDWAREONLY
#warning "Warning:␣Building␣SerialCommand␣without␣SoftwareSerial␣Support"
#endif

#ifndef SERIALCOMMAND_HARDWAREONLY
#include <SoftwareSerial.h>
#endif

#include <string.h>


#define SERIALCOMMANDBUFFER 32
#define MAXSERIALCOMMANDS 10
#define MAXDELIMETER 2

#define SERIALCOMMANDDEBUG 1
#undef SERIALCOMMANDDEBUG // Comment this out to run the library in debug mode (verbose messages)

class SerialCommand
{
	public:
		SerialCommand(); // Constructor
		#ifndef SERIALCOMMAND_HARDWAREONLY
		SerialCommand(SoftwareSerial &SoftSer); // Constructor for using SoftwareSerial objects
		#endif

		void clearBuffer(); // Sets the command buffer to all '\0' (nulls)
		char *next(); // returns pointer to next token found in command buffer (for getting arguments to commands)
		void readSerial(); // Main entry point.
		void addCommand(const char *, void(*)()); // Add commands to processing dictionary
		void addDefaultHandler(void (*function)()); // A handler to call when no valid command received.

	private:
		char inChar; // A character read from the serial stream
		char buffer[SERIALCOMMANDBUFFER]; // Buffer of stored characters while waiting for terminator character
		int bufPos; // Current position in the buffer
		char delim[MAXDELIMETER]; // null-terminated list of character to be used as delimeters for tokenizing (default " ")
		char term; // Character that signals end of command (default '\r')
		char *token; // Returned token from the command buffer as returned by strtok_r
		char *last; // State variable used by strtok_r during processing
		typedef struct _callback {
			char command[SERIALCOMMANDBUFFER];
```

```
                void (*function)();
        } SerialCommandCallback; // Data structure to hold Command/Handler function key−
            value pairs
        int numCommand;
        SerialCommandCallback CommandList[MAXSERIALCOMMANDS]; // Actual
            definition for command/handler array
        void (*defaultHandler)(); // Pointer to the default handler function
        int usingSoftwareSerial; // Used as boolean to see if we're using SoftwareSerial object or
            not
        #ifndef SERIALCOMMAND_HARDWAREONLY
        SoftwareSerial *SoftSerial; // Pointer to a user−created SoftwareSerial object
        #endif
};

#endif //SerialCommand_h
```

Source Code 1.3 SerialCommand.cpp

```
/******************************************************************************
SerialCommand - An Arduino library to tokenize and parse commands received over
a serial port.
Copyright (C) 2011-2013 Steven Cogswell <steven.cogswell@gmail.com>
http://awtfy.com

See SerialCommand.h for version history.

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
*******************************************************************************/

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#include "SerialCommand.h"


#include <string.h>
#ifndef SERIALCOMMAND_HARDWAREONLY
#include <SoftwareSerial.h>
#endif

// Constructor makes sure some things are set.
SerialCommand::SerialCommand()
{
	usingSoftwareSerial=0;
	strncpy(delim," ",MAXDELIMETER); // strtok_r needs a null-terminated string
	term='\r'; // return character, default terminator for commands
	numCommand=0; // Number of callback handlers installed
	clearBuffer();
}

#ifndef SERIALCOMMAND_HARDWAREONLY
// Constructor to use a SoftwareSerial object
SerialCommand::SerialCommand(SoftwareSerial &_SoftSer)
{
        usingSoftwareSerial=1;
        SoftSerial = &_SoftSer;
        strncpy(delim," ",MAXDELIMETER); // strtok_r needs a null−terminated string
        term='\r'; // return character, default terminator for commands
        numCommand=0; // Number of callback handlers installed
        clearBuffer();
}
#endif


//
// Initialize the command buffer being processed to all null characters
//
void SerialCommand::clearBuffer()
{
        for (int i=0; i<SERIALCOMMANDBUFFER; i++)
        {
                buffer[i]='\0';
        }
        bufPos=0;
}

// Retrieve the next token ("word" or "argument") from the Command buffer.
// returns a NULL if no more tokens exist.
char *SerialCommand::next()
{
        char *nextToken;
        nextToken = strtok_r(NULL, delim, &last);
        return nextToken;
}

// This checks the Serial stream for characters, and assembles them into a buffer.
// When the terminator character (default '\r') is seen, it starts parsing the
// buffer for a prefix command, and calls handlers setup by addCommand() member
void SerialCommand::readSerial()
{
        // If we're using the Hardware port, check it. Otherwise check the user−created SoftwareSerial Port
        #ifdef SERIALCOMMAND_HARDWAREONLY
        while (Serial.available() > 0)
        #else
        while ((usingSoftwareSerial==0 && Serial.available() > 0) || (usingSoftwareSerial==1 &&
              SoftSerial−>available() > 0) )
        #endif
        {
                int i;
                boolean matched;
                if (usingSoftwareSerial==0) {
                        // Hardware serial port
                        inChar=Serial.read(); // Read single available character, there may be more
                              waiting
                } else {
                        #ifndef SERIALCOMMAND_HARDWAREONLY
                        // SoftwareSerial port
                        inChar = SoftSerial->read(); // Read single available character, there may be
                              more waiting
                        #endif
                }
                #ifdef SERIALCOMMANDDEBUG
                Serial.print(inChar); // Echo back to serial stream
                #endif
                if (inChar==term) { // Check for the terminator (default '\r') meaning end of command
                        #ifdef SERIALCOMMANDDEBUG
                        Serial.print("Received: ");
                        Serial.println(buffer);
                    #endif
                        bufPos=0; // Reset to start of buffer
                        token = strtok_r(buffer,delim,&last); // Search for command at start of buffer
                        if (token == NULL) return;
                        matched=false;
                        for (i=0; i<numCommand; i++) {
                                #ifdef SERIALCOMMANDDEBUG
                                Serial.print("Comparing [");
                                Serial.print(token);
                                Serial.print("] to [");
                                Serial.print(CommandList[i].command);
                                Serial.println("]");
                                #endif
                                // Compare the found command against the list of known commands for a
                                    match
                                if (strncmp(token,CommandList[i].command,
                                    SERIALCOMMANDBUFFER) == 0)
                                {
                                        #ifdef SERIALCOMMANDDEBUG
                                        Serial.print("Matched Command: ");
                                        Serial.println(token);
                                        #endif
                                        // Execute the stored handler function for the command
                                        (*CommandList[i].function)();
                                        clearBuffer();
                                        matched=true;
                                        break;
                                }
                        }
                        if (matched==false) {
                                (*defaultHandler)();
                                clearBuffer();
                        }

                }
                if (isprint(inChar)) // Only printable characters into the buffer
                {
                        buffer[bufPos++]=inChar; // Put character into buffer
                        buffer[bufPos]='\0'; // Null terminate
                        if (bufPos > SERIALCOMMANDBUFFER-1) bufPos=0; // wrap buffer
                              around if full
                }
        }
}

// Adds a "command" and a handler function to the list of available commands.
// This is used for matching a found token in the buffer, and gives the pointer
// to the handler function to deal with it.
void SerialCommand::addCommand(const char *command, void (*function)())
{
        if (numCommand < MAXSERIALCOMMANDS) {
                #ifdef SERIALCOMMANDDEBUG
                Serial.print(numCommand);
                Serial.print("-");
                Serial.print("Adding command for ");
                Serial.println(command);
                #endif

                strncpy(CommandList[numCommand].command,command,
                     SERIALCOMMANDBUFFER);
                CommandList[numCommand].function = function;
                numCommand++;
        } else {
                // In this case, you tried to push more commands into the buffer than it is compiled to hold.
                // Not much we can do since there is no real visible error assertion, we just ignore adding
                // the command
                #ifdef SERIALCOMMANDDEBUG
                Serial.println("Too many handlers - recompile changing MAXSERIALCOMMANDS
                     ");
                #endif
        }
}

// This sets up a handler to be called in the event that the receveived command string
// isn't in the list of things with handlers.
void SerialCommand::addDefaultHandler(void (*function)())
{
        defaultHandler = function;
}
```

Source Code 1.4 MsTimer2.h

```
#ifndef MsTimer2_h
#define MsTimer2_h

#ifdef __AVR__
#include <avr/interrupt.h>
#elif defined(__arm__) && defined(TEENSYDUINO)
#include <Arduino.h>
#else
#error MsTimer2 library only works on AVR architecture
#endif

namespace MsTimer2 {
	extern unsigned long msecs;
	extern void (*func)();
	extern volatile unsigned long count;
	extern volatile char overflowing;
	extern volatile unsigned int tcnt2;

	void set(unsigned long ms, void (*f)());
	void start();
	void stop();
	void _overflow();
}

#endif
```

Source Code 1.5 MsTimer2.cpp

```
/*
  MsTimer2.h - Using timer2 with 1ms resolution
  Javier Valencia <javiervalencia80@gmail.com>

  https://github.com/PaulStoffregen/MsTimer2

  History:
        6/Jun/14 - V0.7 added support for Teensy 3.0 & 3.1
        29/Dec/11 - V0.6 added support for ATmega32u4, AT90USB646, AT90USB1286 (paul@pjrc.com
            )
                some improvements added by Bill Perry
                note: uses timer4 on Atmega32u4
        29/May/09 - V0.5 added support for Atmega1280 (thanks to Manuel Negri)
        19/Mar/09 - V0.4 added support for ATmega328P (thanks to Jerome Despatis)
        11/Jun/08 - V0.3
                changes to allow working with different CPU frequencies
                added support for ATMega128 (using timer2)
                compatible with ATMega48/88/168/8
        10/May/08 - V0.2 added some security tests and volatile keywords
        9/May/08 - V0.1 released working on ATMEGA168 only


  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
*/

#include <MsTimer2.h>

unsigned long MsTimer2::msecs;
void (*MsTimer2::func)();
volatile unsigned long MsTimer2::count;
volatile char MsTimer2::overflowing;
volatile unsigned int MsTimer2::tcnt2;
#if defined(__arm__) && defined(TEENSYDUINO)
static IntervalTimer itimer;
#endif

void MsTimer2::set(unsigned long ms, void (*f)()) {
        float prescaler = 0.0;

        if (ms == 0)
```

```
			msecs = 1;
		else
			msecs = ms;

		func = f;

#if defined (__AVR_ATmega168__) || defined (__AVR_ATmega48__) || defined (
	__AVR_ATmega88__) || defined (__AVR_ATmega328P__) || defined(
	__AVR_ATmega1280__) || defined(__AVR_ATmega2560__) || defined(
	__AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
		TIMSK2 &= ~(1<<TOIE2);
		TCCR2A &= ~((1<<WGM21) | (1<<WGM20));
		TCCR2B &= ~(1<<WGM22);
		ASSR &= ~(1<<AS2);
		TIMSK2 &= ~(1<<OCIE2A);

		if ((F_CPU >= 1000000UL) && (F_CPU <= 16000000UL)) { // prescaler set to 64
			TCCR2B |= (1<<CS22);
			TCCR2B &= ~((1<<CS21) | (1<<CS20));
			prescaler = 64.0;
		} else if (F_CPU < 1000000UL) { // prescaler set to 8
			TCCR2B |= (1<<CS21);
			TCCR2B &= ~((1<<CS22) | (1<<CS20));
			prescaler = 8.0;
		} else { // F_CPU > 16Mhz, prescaler set to 128
			TCCR2B |= ((1<<CS22) | (1<<CS20));
			TCCR2B &= ~(1<<CS21);
			prescaler = 128.0;
		}
#elif defined (__AVR_ATmega8__)
		TIMSK &= ~(1<<TOIE2);
		TCCR2 &= ~((1<<WGM21) | (1<<WGM20));
		TIMSK &= ~(1<<OCIE2);
		ASSR &= ~(1<<AS2);

		if ((F_CPU >= 1000000UL) && (F_CPU <= 16000000UL)) { // prescaler set to 64
			TCCR2 |= (1<<CS22);
			TCCR2 &= ~((1<<CS21) | (1<<CS20));
			prescaler = 64.0;
		} else if (F_CPU < 1000000UL) { // prescaler set to 8
			TCCR2 |= (1<<CS21);
			TCCR2 &= ~((1<<CS22) | (1<<CS20));
			prescaler = 8.0;
		} else { // F_CPU > 16Mhz, prescaler set to 128
			TCCR2 |= ((1<<CS22) && (1<<CS20));
			TCCR2 &= ~(1<<CS21);
			prescaler = 128.0;
		}
#elif defined (__AVR_ATmega128__)
		TIMSK &= ~(1<<TOIE2);
		TCCR2 &= ~((1<<WGM21) | (1<<WGM20));
		TIMSK &= ~(1<<OCIE2);

	if ((F_CPU >= 1000000UL) && (F_CPU <= 16000000UL)) { // prescaler set to 64
		TCCR2 |= ((1<<CS21) | (1<<CS20));
		TCCR2 &= ~(1<<CS22);
		prescaler = 64.0;
	} else if (F_CPU < 1000000UL) { // prescaler set to 8
		TCCR2 |= (1<<CS21);
		TCCR2 &= ~((1<<CS22) | (1<<CS20));
		prescaler = 8.0;
	} else { // F_CPU > 16Mhz, prescaler set to 256
		TCCR2 |= (1<<CS22);
		TCCR2 &= ~((1<<CS21) | (1<<CS20));
		prescaler = 256.0;
	}
#elif defined (__AVR_ATmega32U4__)
	TCCR4B = 0;
	TCCR4A = 0;
	TCCR4C = 0;
	TCCR4D = 0;
	TCCR4E = 0;
	if (F_CPU >= 16000000L) {
		TCCR4B = (1<<CS43) | (1<<PSR4);
		prescaler = 128.0;
	} else if (F_CPU >= 8000000L) {
		TCCR4B = (1<<CS42) | (1<<CS41) | (1<<CS40) | (1<<PSR4);
		prescaler = 64.0;
	} else if (F_CPU >= 4000000L) {
		TCCR4B = (1<<CS42) | (1<<CS41) | (1<<PSR4);
		prescaler = 32.0;
	} else if (F_CPU >= 2000000L) {
		TCCR4B = (1<<CS42) | (1<<CS40) | (1<<PSR4);
		prescaler = 16.0;
	} else if (F_CPU >= 1000000L) {
		TCCR4B = (1<<CS42) | (1<<PSR4);
		prescaler = 8.0;
	} else if (F_CPU >= 500000L) {
		TCCR4B = (1<<CS41) | (1<<CS40) | (1<<PSR4);
		prescaler = 4.0;
	} else {
		TCCR4B = (1<<CS41) | (1<<PSR4);
		prescaler = 2.0;
	}
	tcnt2 = (int)((float)F_CPU * 0.001 / prescaler) - 1;
	OCR4C = tcnt2;
	return;
#elif defined(__arm__) && defined(TEENSYDUINO)
	// nothing needed here
#else
#error Unsupported CPU type
#endif

	tcnt2 = 256 - (int)((float)F_CPU * 0.001 / prescaler);
}

```

```
void MsTimer2::start() {
	count = 0;
	overflowing = 0;
#if defined (__AVR_ATmega168__) || defined (__AVR_ATmega48__) || defined (
    __AVR_ATmega88__) || defined (__AVR_ATmega328P__) || defined (
    __AVR_ATmega1280__) || defined(__AVR_ATmega2560__) || defined(
    __AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
	TCNT2 = tcnt2;
	TIMSK2 |= (1<<TOIE2);
#elif defined (__AVR_ATmega128__)
	TCNT2 = tcnt2;
	TIMSK |= (1<<TOIE2);
#elif defined (__AVR_ATmega8__)
	TCNT2 = tcnt2;
	TIMSK |= (1<<TOIE2);
#elif defined (__AVR_ATmega32U4__)
	TIFR4 = (1<<TOV4);
	TCNT4 = 0;
	TIMSK4 = (1<<TOIE4);
#elif defined(__arm__) && defined(TEENSYDUINO)
	itimer.begin(MsTimer2::_overflow, 1000);
#endif
}

void MsTimer2::stop() {
#if defined (__AVR_ATmega168__) || defined (__AVR_ATmega48__) || defined (
    __AVR_ATmega88__) || defined (__AVR_ATmega328P__) || defined (
    __AVR_ATmega1280__) || defined(__AVR_ATmega2560__) || defined(
    __AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
	TIMSK2 &= ~(1<<TOIE2);
#elif defined (__AVR_ATmega128__)
	TIMSK &= ~(1<<TOIE2);
#elif defined (__AVR_ATmega8__)
	TIMSK &= ~(1<<TOIE2);
#elif defined (__AVR_ATmega32U4__)
	TIMSK4 = 0;
#elif defined(__arm__) && defined(TEENSYDUINO)
	itimer.end();
#endif
}

void MsTimer2::_overflow() {
	count += 1;

	if (count >= msecs && !overflowing) {
		overflowing = 1;
		count = count - msecs; // subtract ms to catch missed overflows
					// set to 0 if you don't want this.
		(*func)();
		overflowing = 0;
	}
}

```

```
#if defined (__AVR__)
#if defined (__AVR_ATmega32U4__)
ISR(TIMER4_OVF_vect) {
#else
ISR(TIMER2_OVF_vect) {
#endif
#if defined (__AVR_ATmega168__) || defined (__AVR_ATmega48__) || defined (
    __AVR_ATmega88__) || defined (__AVR_ATmega328P__) || defined (
    __AVR_ATmega1280__) || defined(__AVR_ATmega2560__) || defined(
    __AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
        TCNT2 = MsTimer2::tcnt2;
#elif defined (__AVR_ATmega128__)
        TCNT2 = MsTimer2::tcnt2;
#elif defined (__AVR_ATmega8__)
        TCNT2 = MsTimer2::tcnt2;
#elif defined (__AVR_ATmega32U4__)
        // not necessary on 32u4's high speed timer4
#endif
        MsTimer2::_overflow();
}
#endif // AVR
```

1.1.2 esp32(Web サーバ) 向けソース

Source Code 1.6 esp32_firm.ino

```
#include <Time.h>
#include <TimeLib.h>
#include <WiFi.h>
#include <Wire.h>

#include "RTClib.h"
RTC_DS3231 rtc;

#include <SSCI_BME280.h>
SSCI_BME280 bme280;
uint8_t i2c_addr = 0x77;

#include "ESPAsyncWebServer.h"
#include <TinyGPS.h>

// Timer Interrupt setting
hw_timer_t * timer = NULL;

volatile SemaphoreHandle_t timerSemaphore;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

const char* ssid = "yurucamp";
const char* pass = "mokemoke";

const int8_t timezone = 9;
unsigned int dpmode = 0;

AsyncWebServer server(80);

const int SW3 = 14;
const int SW4 = 27;
const int SW5 = 26;

volatile char func_btn[2] = {
  0,
  0,
};

// nixie tube disolay num
// 0-9:そのまま
// 10: display none
volatile int display_num[8] = {
  0,
  1,
  2,
  3,
  4,
  5,
  6,
  7,
};

// dot none:0
// only right dot:1
// only left dot:2
// right and left dot:3
volatile int display_dot[8] = {
  0,
  1,
  0,
  0,
  1,
  0,
  0,
  0,
};

volatile bool func_enable[10] = {
  true,
  true,
  true,
  true,
  true,
  true,
  true,
  true,
  true,
  true,
};


String html ="<!DOCTYPE html> <html lang=\"ja\"> <head> <meta charset=\"utf-8\"> <meta http-equiv=\"X-UA-Compatible\" content=\"IE=edge\"> <meta name=\"viewport\" content=\"width=device-width, initial-scale=1\"> <title>Bootstrap Sample</title> <link href=\"https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css\" rel=\"stylesheet\"> <script src=\"https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js\"></script> <script src=\"https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js\"></script> <script src=\"https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js\"></script> </head> <body> <header style=\"background-color:white\"></header> <form> <div class=\"container-fluid\"> <div class=\"row\"> <div class=\"container\"> <h3>ボタンのモード設定</h3> <table class=\"table table-bordered\"> <thead> <tr> <th>ボタンの名前</th> <th>モード</th> </tr> </thead> <tbody> <tr> <th scope=\"row\">func1</th> <td> <select name=\"func1\"> <option value=\"0\">日付</option> <option value=\"1\">時刻</option> <option value=\"2\">気圧</option> <option value=\"3\">気温、湿度</option> <option value=\"4\">緯度経度</option> <option value=\"5\">APIモード</option> <option value=\"6\">タイマー</option> </select> </td> </tr> <tr> <th scope=\"row\">func2</th> <td> <select name=\"func2\"> <option value=\"0\">日付</option> <option value=\"1\">時刻</option> <option value=\"2\">気圧</option> <option value=\"3\">気温、湿度</option> <option value=\"4\">緯度経度</option> <option value=\"5\">APIモード</option> <option value=\"6\">タイマー</option> </select> </td> </tr> </tbody> </table> </div> <br> <div class=\"container\"><br> <h3>Wifi設定</h3> <p>現在のIPアドレスは</p> <!-- Wifi情報をとってくる --> <p>現在接続中の
```

```
    Wifi は</p> <!−− Wifi 情報をとってくる −−> <br> <div class=\"form−group\"> <h3>
    NTP アドレス</h3> <input type=\"email\" class=\"form−control\" id=\"exampleInputaddr\"
    placeholder=\"example\"> </div> </form> <br> <div class=\"checkbox\"> <h3>有効にするモー
    ド</h3> <label> <input type=\"checkbox\" name=\"date\"> 日付<br> <input type=\"checkbox\"
    name=\"clock\"> 時刻<br> <input type=\"checkbox\" name=\"pressure\"> 気圧<br> <input type
    =\"checkbox\" name=\"temp\"> 気温、湿度<br> <input type=\"checkbox\" name=\"gps\"> 緯度
    、経度<br> <input type=\"checkbox\" name=\"api\"> API モード<br> <input type=\"checkbox\"
    name=\"timer\"> タイマー<br> </label> </div> <button type=\"submit\" class=\"btn btn−
    primary\">送信</button> </div> </div> </div> </form> <footer style=\"background−color:white
    \"></footer> </body> </html>";

HardwareSerial serial0(0);
HardwareSerial atmega_serial(2);

void adjustByNTP(){
  configTzTime("JST-9", "ntp.nict.jp", "time.google.com", "ntp.jst.mfeed.ad.jp");
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed␣to␣obtain␣time");
  }
  rtc.adjust(DateTime(time(NULL))+TimeSpan(0, timezone, 0, 0));
}

void setNum(){
  String command = "num␣";

  for(int n = 0; n < 8; n++){
    String temp = String(command + String(display_num[n]));
    command = temp;
  }

  //for debug
  // serial0.println(command);
  atmega_serial.println(command);
}

void setDot(){
  String command = "dot␣";

  for(int n = 0; n < 8; n++){
    String temp = String(command + String(display_dot[n]));
    command = temp;
  }

  //for debug
  // serial0.println(command);
  atmega_serial.println(command);
}

void setup() {
  pinMode(4,OUTPUT);
  pinMode(19,OUTPUT);
  pinMode(18,OUTPUT);

  // speaker high
  digitalWrite(4,HIGH);

  //button setting
  pinMode(SW3,INPUT);
  pinMode(SW4,INPUT);
  pinMode(SW5,INPUT_PULLUP);

  delay(5000);

  serial0.begin(115200);

  // open arduino serial
  // rx:19 tx:18
  atmega_serial.begin(9600, SERIAL_8N1, 19, 18);

  setNum();
  setDot();

  /*
  // for test
  atmega_serial.println("dcdc_on");
  atmega_serial.println("num 11451419");
  */

  bool isWiFiConnected = true;

  serial0.printf("Connecting to %s ", ssid);
  WiFi.disconnect(true);
  WiFi.begin(ssid, pass);
  unsigned long time = millis();
  while (WiFi.status() != WL_CONNECTED) {
    delay(100);
    serial0.print(".");

    if(millis()-time>10000){
      serial0.print("Can't connect Wi-Fi");
      WiFi.disconnect(true);
      isWiFiConnected = false;
      break;
    }
  }
  serial0.println(" CONNECTED");

  //RTC のあれこれ
  if (! rtc.begin()) {
    serial0.println("Couldn't find RTC");
    while (1);
  }

  if(isWiFiConnected){
    adjustByNTP();
```

```
}

//bme280 set up
uint8_t osrs_t = 1; //Temperature oversampling x 1
uint8_t osrs_p = 1; //Pressure oversampling x 1
uint8_t osrs_h = 1; //Humidity oversampling x 1
uint8_t bme280mode = 3; //Normal mode
uint8_t t_sb = 5; //Tstandby 1000ms
uint8_t filter = 0; //Filter off
uint8_t spi3w_en = 0; //3-wire SPI Disable

bme280.setMode(i2c_addr, osrs_t, osrs_p, osrs_h, bme280mode, t_sb, filter, spi3w_en);
bme280.readTrim();

server.on("/setting", HTTP_GET, [&](AsyncWebServerRequest *request){

  int paramsNr = request->params();
  serial0.println(paramsNr);

  for(int i=0;i<paramsNr;i++){
    AsyncWebParameter* p = request->getParam(i);

    if(p->name() == "num" && dpmode == 4){
      String param = p->value();

      for(char i = 0; i < 8; i++){
        if(param[i] == ':'){
          // display_pattern[i][0] = 0;
          // display_pattern[i][1] = 0;
          continue;
        }

        // memcpy(display_pattern[i], (void*)num_signal_pattern[param[i] - '0'], 2);

      }
    }

    if(p->name() == "dot" && dpmode == 4){
      String param = p->value();

      for(char i = 0; i < 8; ++i){
        // display_pattern[i][0] &= 0b11111100;

        switch(param[i] - '0'){
          case 0:
            // display_pattern[i][0] |= dot_signal_pattern[0][0];
            break;
          case 1:
            // display_pattern[i][0] |= dot_signal_pattern[1][0];
            break;
          case 2:
            // display_pattern[i][0] |= dot_signal_pattern[2][0];
            break;
```

```
                case 3:
                    // display_pattern[i][0]  |= dot_signal_pattern[3][0];
                    break;
            }
        }
    }

    if(p->name() == "mode"){
        dpmode = p->value()[0] - '0';
    }
  }

  request->send(200, "text/html", "<p>message received</p>");
});

server.on("/", HTTP_GET, [&](AsyncWebServerRequest *request){
  int params_num = request->params();

  bool flag = false;
  bool temp_func_enable[10] = {
    false,
    false,
    false,
    false,
    false,
    false,
    false,
    false,
    false,
    false,
  };

  for(int i=0; i < params_num; ++i){
    AsyncWebParameter* p = request->getParam(i);

    if(p->name() == "func1"){
      func_btn[0] = p->value()[0] - '0';
    }

    if(p->name() == "func2"){
      func_btn[1] = p->value()[0] - '0';
    }

    if(p->name() == "clock"){
      flag = true;
      temp_func_enable[0] = true;
    }

    if(p->name() == "date"){
      flag = true;
      temp_func_enable[1] = true;
    }

```

```
      if(p->name() == "temp"){
        flag = true;
        temp_func_enable[2] = true;
      }

      if(p->name() == "pressure"){
        flag = true;
        temp_func_enable[3] = true;
      }

      if(p->name() == "api"){
        flag = true;
        temp_func_enable[4] = true;
      }

      if(p->name() == "gps"){
        flag = true;
        temp_func_enable[5] = true;
      }

      if(p->name() == "timer"){
        flag = true;
        temp_func_enable[6] = true;
      }

    }

    if(flag){
      for(char i = 0; i < 10; ++i){
        func_enable[i] = temp_func_enable[i];
      }
    }

    request->send(200, "text/html", html);
  });

  server.begin();
}

void setDisplayTime(DateTime now){
  /*
  memcpy(display_pattern[0], (void*)num_signal_pattern[now.hour()%100/10], 2);
  memcpy(display_pattern[1], (void*)num_signal_pattern[now.hour()%10], 2);
  memcpy(display_pattern[2], (void*)dot_signal_pattern[2], 2);
  memcpy(display_pattern[3], (void*)num_signal_pattern[now.minute()%100/10], 2);
  memcpy(display_pattern[4], (void*)num_signal_pattern[now.minute()%10], 2);
  memcpy(display_pattern[5], (void*)dot_signal_pattern[2], 2);
  memcpy(display_pattern[6], (void*)num_signal_pattern[now.second()%100/10], 2);
  memcpy(display_pattern[7], (void*)num_signal_pattern[now.second()%10], 2);
  */
}

void setDisplayDate(DateTime now){
}

void setDisplayThermoHumidity(double temperature, double humidity){

}

void setDisplayPressure(double pressure){
}

double temp_act, press_act, hum_act; //最終的に表示される値を入れる変数

void loop() {

}
```

Source Code 1.7 NTPClient.h

```
#pragma once

#include "Arduino.h"

#include <Udp.h>

#define SEVENZYYEARS 2208988800UL
#define NTP_PACKET_SIZE 48
#define NTP_DEFAULT_LOCAL_PORT 1337

class NTPClient {
  private:
    UDP* _udp;
    bool _udpSetup = false;

    const char* _poolServerName = "pool.ntp.org"; // Default time server
    int _port = NTP_DEFAULT_LOCAL_PORT;
    long _timeOffset = 0;

    unsigned long _updateInterval = 60000; // In ms

    unsigned long _currentEpoc = 0; // In s
    unsigned long _lastUpdate = 0; // In ms

    byte _packetBuffer[NTP_PACKET_SIZE];

    void sendNTPPacket();

  public:
    NTPClient(UDP& udp);
    NTPClient(UDP& udp, long timeOffset);
    NTPClient(UDP& udp, const char* poolServerName);
    NTPClient(UDP& udp, const char* poolServerName, long timeOffset);
    NTPClient(UDP& udp, const char* poolServerName, long timeOffset, unsigned long
         updateInterval);

    /**
     * Set time server name
     *
     * @param poolServerName
     */
    void setPoolServerName(const char* poolServerName);

    /**
     * Starts the underlying UDP client with the default local port
     */
    void begin();

    /**
     * Starts the underlying UDP client with the specified local port
     */
    void begin(int port);

    /**
      * This should be called in the main loop of your application. By default an update from the NTP Server
            is only
      * made every 60 seconds. This can be configured in the NTPClient constructor.
      *
      * @return true on success, false on failure
      */
    bool update();

    /**
      * This will force the update from the NTP Server.
      *
      * @return true on success, false on failure
      */
    bool forceUpdate();

    int getDay() const;
    int getHours() const;
    int getMinutes() const;
    int getSeconds() const;

    /**
      * Changes the time offset. Useful for changing timezones dynamically
      */
    void setTimeOffset(int timeOffset);

    /**
      * Set the update interval to another frequency. E.g. useful when the
      * timeOffset should not be set in the constructor
      */
    void setUpdateInterval(unsigned long updateInterval);

    /**
      * @return time formatted like 'hh:mm:ss'
      */
    String getFormattedTime() const;

    /**
      * @return time in seconds since Jan. 1, 1970
      */
    unsigned long getEpochTime() const;

    /**
      * Stops the underlying UDP client
      */
    void end();
};
```

Source Code 1.8 NTPClient.cpp

```
/**
 * The MIT License (MIT)
 * Copyright (c) 2015 by Fabrice Weinberg
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
      OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
      THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
      FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
      IN THE
 * SOFTWARE.
 */

#include "NTPClient.h"

NTPClient::NTPClient(UDP& udp) {
  this->_udp = &udp;
}

NTPClient::NTPClient(UDP& udp, long timeOffset) {
  this->_udp = &udp;
  this->_timeOffset = timeOffset;
}

NTPClient::NTPClient(UDP& udp, const char* poolServerName) {
  this->_udp = &udp;
  this->_poolServerName = poolServerName;
}

NTPClient::NTPClient(UDP& udp, const char* poolServerName, long timeOffset) {
  this->_udp = &udp;
  this->_timeOffset = timeOffset;
  this->_poolServerName = poolServerName;
}

NTPClient::NTPClient(UDP& udp, const char* poolServerName, long timeOffset, unsigned long
      updateInterval) {
  this->_udp = &udp;
  this->_timeOffset = timeOffset;
  this->_poolServerName = poolServerName;
  this->_updateInterval = updateInterval;
}

void NTPClient::begin() {
  this->begin(NTP_DEFAULT_LOCAL_PORT);
}

void NTPClient::begin(int port) {
  this->_port = port;

  this->_udp->begin(this->_port);

  this->_udpSetup = true;
}

bool NTPClient::forceUpdate() {
  #ifdef DEBUG_NTPClient
    Serial.println("Update from NTP Server");
  #endif

  this->sendNTPPacket();

  // Wait till data is there or timeout...
  byte timeout = 0;
  int cb = 0;
  do {
    delay ( 10 );
    cb = this->_udp->parsePacket();
    if (timeout > 100) return false; // timeout after 1000 ms
    timeout++;
  } while (cb == 0);

  this->_lastUpdate = millis() - (10 * (timeout + 1)); // Account for delay in reading the time

  this->_udp->read(this->_packetBuffer, NTP_PACKET_SIZE);

  unsigned long highWord = word(this->_packetBuffer[40], this->_packetBuffer[41]);
  unsigned long lowWord = word(this->_packetBuffer[42], this->_packetBuffer[43]);
  // combine the four bytes (two words) into a long integer
  // this is NTP time (seconds since Jan 1 1900):
  unsigned long secsSince1900 = highWord << 16 | lowWord;

  this->_currentEpoc = secsSince1900 - SEVENZYYEARS;

  return true;
}

bool NTPClient::update() {
  if ((millis() - this->_lastUpdate >= this->_updateInterval) // Update after _updateInterval
    || this->_lastUpdate == 0) { // Update if there was no update yet.
    if (!this->_udpSetup) this->begin(); // setup the UDP client if needed
    return this->forceUpdate();
  }
  return true;
}

unsigned long NTPClient::getEpochTime() const {
  return this->_timeOffset + // User offset
         this->_currentEpoc + // Epoc returned by the NTP server
         ((millis() - this->_lastUpdate) / 1000); // Time since last update
}

int NTPClient::getDay() const {
  return (((this->getEpochTime() / 86400L) + 4 ) % 7); //0 is Sunday
}
int NTPClient::getHours() const {
  return ((this->getEpochTime() % 86400L) / 3600);
}
int NTPClient::getMinutes() const {
  return ((this->getEpochTime() % 3600) / 60);
}
int NTPClient::getSeconds() const {
  return (this->getEpochTime() % 60);
}

String NTPClient::getFormattedTime() const {
  unsigned long rawTime = this->getEpochTime();
  unsigned long hours = (rawTime % 86400L) / 3600;
  String hoursStr = hours < 10 ? "0" + String(hours) : String(hours);

  unsigned long minutes = (rawTime % 3600) / 60;
  String minuteStr = minutes < 10 ? "0" + String(minutes) : String(minutes);

  unsigned long seconds = rawTime % 60;
  String secondStr = seconds < 10 ? "0" + String(seconds) : String(seconds);

  return hoursStr + ":" + minuteStr + ":" + secondStr;
}

void NTPClient::end() {
  this->_udp->stop();

  this->_udpSetup = false;
}

void NTPClient::setTimeOffset(int timeOffset) {
  this->_timeOffset = timeOffset;
}

void NTPClient::setUpdateInterval(unsigned long updateInterval) {
  this->_updateInterval = updateInterval;
}

void NTPClient::setPoolServerName(const char* poolServerName) {
    this->_poolServerName = poolServerName;
}
```

```

void NTPClient::sendNTPPacket() {
  // set all bytes in the buffer to 0
  memset(this->_packetBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  this->_packetBuffer[0] = 0b11100011; // LI, Version, Mode
  this->_packetBuffer[1] = 0; // Stratum, or type of clock
  this->_packetBuffer[2] = 6; // Polling Interval
  this->_packetBuffer[3] = 0xEC; // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  this->_packetBuffer[12] = 49;
  this->_packetBuffer[13] = 0x4E;
  this->_packetBuffer[14] = 49;
  this->_packetBuffer[15] = 52;

  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  this->_udp->beginPacket(this->_poolServerName, 123); //NTP requests are to port 123
  this->_udp->write(this->_packetBuffer, NTP_PACKET_SIZE);
  this->_udp->endPacket();
}
```

Source Code 1.9 SSCI_BME280.h

```
/*****************************************************
  BME280 liblary for Arduino

  https://www.switch-science.com/catalog/2236/
  https://www.switch-science.com/catalog/2323/

*****************************************************/

#if (ARDUINO >= 100)
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

// I2C Address
//#define BME280_ADDRESS 0x76

// BME280 Registers
#define BME280_REG_calib00 0x88
#define BME280_REG_calib25 0xa1
#define BME280_REG_ID 0xd0
#define BME280_REG_reset 0xe0
#define BME280_REG_calib26 0xe1
#define BME280_REG_ctrl_hum 0xf2
#define BME280_REG_status 0xf3
#define BME280_REG_ctrl_meas 0xf4
#define BME280_REG_config 0xf5
#define BME280_REG_press_msb 0xf7
#define BME280_REG_press_lsb 0xf8
#define BME280_REG_press_xlsb 0xf9
#define BME280_REG_temp_msb 0xfa
#define BME280_REG_temp_lsb 0xfb
#define BME280_REG_temp_xlsb 0xfc
#define BME280_REG_hum_msb 0xfd
#define BME280_REG_hum_lsb 0xfe

// Caribration data storage
typedef struct {
  uint16_t dig_T1;
  int16_t dig_T2;
  int16_t dig_T3;
  uint16_t dig_P1;
  int16_t dig_P2;
  int16_t dig_P3;
  int16_t dig_P4;
  int16_t dig_P5;
  int16_t dig_P6;
  int16_t dig_P7;
  int16_t dig_P8;
  int16_t dig_P9;
  int8_t dig_H1;
  int16_t dig_H2;
  int8_t dig_H3;
  int16_t dig_H4;
  int16_t dig_H5;
  int8_t dig_H6;
} BME280_calib_data;


class SSCI_BME280 {
  public:
    SSCI_BME280();
    void setMode(
      uint8_t i2c_addr, //I2C Address
      uint8_t osrs_t, //Temperature oversampling
      uint8_t osrs_p, //Pressure oversampling
      uint8_t osrs_h, //Humidity oversampling
      uint8_t bme280mode, //Mode Sleep/Forced/Normal
      uint8_t t_sb, //Tstandby
      uint8_t filter, //Filter off
      uint8_t spi3w_en //3-wire SPI Enable/Disable
    );
    void readTrim();
    void readData(double *temp_act, double *press_act, double *hum_act);

  private:
    signed long int calibration_T(signed long int adc_T);
    unsigned long int calibration_P(signed long int adc_P);
    unsigned long int calibration_H(signed long int adc_H);
    void writeReg(uint8_t reg_address, uint8_t data);
    signed long int t_fine;
    int _i2c_addr;
    BME280_calib_data calibData;
};
```

Source Code 1.10 SSCI_BME280.cpp

```
#include <Wire.h>
#include "SSCI_BME280.h"

void SSCI_BME280::writeReg(uint8_t reg_address, uint8_t data)
{
  Wire.beginTransmission(_i2c_addr);
  Wire.write(reg_address);
  Wire.write(data);
  Wire.endTransmission();
}

SSCI_BME280::SSCI_BME280() {

}

void SSCI_BME280::setMode(uint8_t i2c_addr, uint8_t osrs_t, uint8_t osrs_p, uint8_t osrs_h,
      uint8_t bme280mode, uint8_t t_sb, uint8_t filter, uint8_t spi3w_en) {
  uint8_t ctrl_meas_reg = (osrs_t << 5) | (osrs_p << 2) | bme280mode;
  uint8_t config_reg = (t_sb << 5) | (filter << 2) | spi3w_en;
  uint8_t ctrl_hum_reg = osrs_h;
  _i2c_addr = i2c_addr;
  writeReg(BME280_REG_ctrl_hum, ctrl_hum_reg);
  writeReg(BME280_REG_ctrl_meas, ctrl_meas_reg);
  writeReg(BME280_REG_config, config_reg);
}

void SSCI_BME280::readTrim()
{
  uint8_t data[33], i = 0;
  Wire.beginTransmission(_i2c_addr);
  Wire.write(BME280_REG_calib00);
  Wire.endTransmission();
  Wire.requestFrom(_i2c_addr, 24);
  while (Wire.available()) {
    data[i] = Wire.read();
    i++;
  }

  Wire.beginTransmission(_i2c_addr);
  Wire.write(BME280_REG_calib25);
  Wire.endTransmission();
  Wire.requestFrom(_i2c_addr, 1);
  data[i] = Wire.read();
  i++;

  Wire.beginTransmission(_i2c_addr);
  Wire.write(BME280_REG_calib26);
  Wire.endTransmission();
  Wire.requestFrom(_i2c_addr, 8);
  while (Wire.available()) {
    data[i] = Wire.read();
    i++;
  }
  calibData.dig_T1 = (data[1] << 8) | data[0];
  calibData.dig_T2 = (data[3] << 8) | data[2];
  calibData.dig_T3 = (data[5] << 8) | data[4];
  calibData.dig_P1 = (data[7] << 8) | data[6];
  calibData.dig_P2 = (data[9] << 8) | data[8];
  calibData.dig_P3 = (data[11] << 8) | data[10];
  calibData.dig_P4 = (data[13] << 8) | data[12];
  calibData.dig_P5 = (data[15] << 8) | data[14];
  calibData.dig_P6 = (data[17] << 8) | data[16];
  calibData.dig_P7 = (data[19] << 8) | data[18];
  calibData.dig_P8 = (data[21] << 8) | data[20];
  calibData.dig_P9 = (data[23] << 8) | data[22];
  calibData.dig_H1 = data[24];
  calibData.dig_H2 = (data[26] << 8) | data[25];
  calibData.dig_H3 = data[27];
  calibData.dig_H4 = (data[28] << 4) | (0x0F & data[29]);
  calibData.dig_H5 = (data[30] << 4) | ((data[29] >> 4) & 0x0F);
  calibData.dig_H6 = data[31];
}

void SSCI_BME280::readData(double *temp_act, double *press_act, double *hum_act)
{
  int i = 0;
  uint32_t data[8];
  unsigned long int hum_raw, temp_raw, press_raw;

  Wire.beginTransmission(_i2c_addr);
  Wire.write(BME280_REG_press_msb);
  Wire.endTransmission();
  Wire.requestFrom(_i2c_addr, 8);
  while (Wire.available()) {
    data[i] = Wire.read();
    i++;
  }
  press_raw = (data[0] << 12) | (data[1] << 4) | (data[2] >> 4);
  temp_raw = (data[3] << 12) | (data[4] << 4) | (data[5] >> 4);
  hum_raw = (data[6] << 8) | data[7];
  *temp_act = (double)calibration_T(temp_raw) / 100.0;
  *press_act = (double)calibration_P(press_raw) / 100.0;
  *hum_act = (double)calibration_H(hum_raw) / 1024.0;
}


signed long int SSCI_BME280::calibration_T(signed long int adc_T)
{

  signed long int var1, var2, T;
  var1 = ((((adc_T >> 3) - ((signed long int)calibData.dig_T1 << 1))) * ((signed long int)calibData.
        dig_T2)) >> 11;
  var2 = (((((adc_T >> 4) - ((signed long int)calibData.dig_T1)) * ((adc_T >> 4) - ((signed long int
        )calibData.dig_T1))) >> 12) * ((signed long int)calibData.dig_T3)) >> 14;

  t_fine = var1 + var2;
  T = (t_fine * 5 + 128) >> 8;
  return T;
}
unsigned long int SSCI_BME280::calibration_P(signed long int adc_P)
{
  signed long int var1, var2;
  unsigned long int P;
  var1 = (((signed long int)t_fine) >> 1) - (signed long int)64000;
  var2 = (((var1 >> 2) * (var1 >> 2)) >> 11) * ((signed long int)calibData.dig_P6);
  var2 = var2 + ((var1 * ((signed long int)calibData.dig_P5)) << 1);
  var2 = (var2 >> 2) + (((signed long int)calibData.dig_P4) << 16);
  var1 = (((calibData.dig_P3 * (((var1 >> 2) * (var1 >> 2)) >> 13)) >> 3) + ((((signed long int)
        calibData.dig_P2) * var1) >> 1)) >> 18;
  var1 = ((((32768 + var1)) * ((signed long int)calibData.dig_P1)) >> 15);
  if (var1 == 0)
  {
    return 0;
  }
  P = (((unsigned long int)(((signed long int)1048576) - adc_P) - (var2 >> 12))) * 3125;
  if (P < 0x80000000)
  {
    P = (P << 1) / ((unsigned long int) var1);
  }
  else
  {
    P = (P / (unsigned long int)var1) * 2;
  }
  var1 = (((signed long int)calibData.dig_P9) * ((signed long int)(((P >> 3) * (P >> 3)) >> 13))) >>
        12;
  var2 = (((signed long int)(P >> 2)) * ((signed long int)calibData.dig_P8)) >> 13;
  P = (unsigned long int)((signed long int)P + ((var1 + var2 + calibData.dig_P7) >> 4));
  return P;
}

unsigned long int SSCI_BME280::calibration_H(signed long int adc_H)
{
  signed long int v_x1;

  v_x1 = (t_fine - ((signed long int)76800));
  v_x1 = (((((adc_H << 14) - (((signed long int)calibData.dig_H4) << 20) - (((signed long int)
        calibData.dig_H5) * v_x1)) +
            ((signed long int)16384)) >> 15) * (((((((v_x1 * ((signed long int)calibData.dig_H6))
                >> 10) *
                (((v_x1 * ((signed long int)calibData.dig_H3)) >> 11) + ((signed long int) 32768)))
                    >> 10) + ((signed long int)2097152)) *
                ((signed long int)calibData.dig_H2) + 8192) >> 14));
  v_x1 = (v_x1 - (((((v_x1 >> 15) * (v_x1 >> 15)) >> 7) * ((signed long int)calibData.dig_H1)) >>
        4));
  v_x1 = (v_x1 < 0 ? 0 : v_x1);
  v_x1 = (v_x1 > 419430400 ? 419430400 : v_x1);
  return (unsigned long int)(v_x1 >> 12);
}
```

Source Code 1.11 TImeLib.h

```
/*
  time.h - low level time and date functions
*/

/*
  July 3 2011 - fixed elapsedSecsThisWeek macro (thanks Vincent Valdy for this)
              - fixed daysToTime_t macro (thanks maniacbug)
*/

#ifndef _Time_h
#ifdef __cplusplus
#define _Time_h

#include <inttypes.h>
#ifndef __AVR__
#include <sys/types.h> // for __time_t_defined, but avr libc lacks sys/types.h
#endif


#if !defined(__time_t_defined) // avoid conflict with newlib or other posix libc
typedef unsigned long time_t;
#endif


// This ugly hack allows us to define C++ overloaded functions, when included
// from within an extern "C", as newlib's sys/stat.h does. Actually it is
// intended to include "time.h" from the C library (on ARM, but AVR does not
// have that file at all). On Mac and Windows, the compiler will find this
// "Time.h" instead of the C library "time.h", so we may cause other weird
// and unpredictable effects by conflicting with the C library header "time.h",
// but at least this hack lets us define C++ functions as intended. Hopefully
// nothing too terrible will result from overriding the C library header?!
extern "C++" {
typedef enum {timeNotSet, timeNeedsSync, timeSet
} timeStatus_t ;

typedef enum {
    dowInvalid, dowSunday, dowMonday, dowTuesday, dowWednesday, dowThursday,
        dowFriday, dowSaturday
} timeDayOfWeek_t;

typedef enum {
    tmSecond, tmMinute, tmHour, tmWday, tmDay,tmMonth, tmYear, tmNbrFields
} tmByteFields;

typedef struct {
  uint8_t Second;
  uint8_t Minute;
  uint8_t Hour;
  uint8_t Wday; // day of week, sunday is day 1
  uint8_t Day;
  uint8_t Month;
  uint8_t Year; // offset from 1970;
} tmElements_t, TimeElements, *tmElementsPtr_t;

//convenience macros to convert to and from tm years
#define tmYearToCalendar(Y) ((Y) + 1970) // full four digit year
#define CalendarYrToTm(Y) ((Y) − 1970)
#define tmYearToY2k(Y) ((Y) − 30) // offset is from 2000
#define y2kYearToTm(Y) ((Y) + 30)

typedef time_t(*getExternalTime)();
//typedef void (*setExternalTime)(const time_t); // not used in this version


/*
    ==============================================================================
    */
/* Useful Constants */
#define SECS_PER_MIN ((time_t)(60UL))
#define SECS_PER_HOUR ((time_t)(3600UL))
#define SECS_PER_DAY ((time_t)(SECS_PER_HOUR * 24UL))
#define DAYS_PER_WEEK ((time_t)(7UL))
#define SECS_PER_WEEK ((time_t)(SECS_PER_DAY * DAYS_PER_WEEK))
#define SECS_PER_YEAR ((time_t)(SECS_PER_DAY * 365UL)) // TODO: ought to handle leap years
#define SECS_YR_2000 ((time_t)(946684800UL)) // the time at the start of y2k

/* Useful Macros for getting elapsed time */
#define numberOfSeconds(_time_) ((_time_) % SECS_PER_MIN)
#define numberOfMinutes(_time_) (((_time_) / SECS_PER_MIN) % SECS_PER_MIN)
#define numberOfHours(_time_) (((_time_) % SECS_PER_DAY) / SECS_PER_HOUR)
#define dayOfWeek(_time_) ((((_time_) / SECS_PER_DAY + 4) % DAYS_PER_WEEK)+1) // 1 =
    Sunday
#define elapsedDays(_time_) ((_time_) / SECS_PER_DAY) // this is number of days since Jan 1 1970
#define elapsedSecsToday(_time_) ((_time_) % SECS_PER_DAY) // the number of seconds since last
    midnight
// The following macros are used in calculating alarms and assume the clock is set to a date later than Jan 1
    1971
// Always set the correct time before settting alarms
#define previousMidnight(_time_) (((_time_) / SECS_PER_DAY) * SECS_PER_DAY) // time at the
    start of the given day
#define nextMidnight(_time_) (previousMidnight(_time_) + SECS_PER_DAY) // time at the end of
    the given day
#define elapsedSecsThisWeek(_time_) (elapsedSecsToday(_time_) + ((dayOfWeek(_time_)−1) *
    SECS_PER_DAY)) // note that week starts on day 1
#define previousSunday(_time_) ((_time_) − elapsedSecsThisWeek(_time_)) // time at the start of the
    week for the given time
#define nextSunday(_time_) (previousSunday(_time_)+SECS_PER_WEEK) // time at the end of the
    week for the given time


/* Useful Macros for converting elapsed time to a time_t */
#define minutesToTime_t ((M)) ( (M) * SECS_PER_MIN)
#define hoursToTime_t ((H)) ( (H) * SECS_PER_HOUR)
#define daysToTime_t ((D)) ( (D) * SECS_PER_DAY) // fixed on Jul 22 2011
```

```
#define weeksToTime_t ((W)) ( (W) * SECS_PER_WEEK)

/* ==============================================================================
*/
/* time and date functions */
int hour(); // the hour now
int hour(time_t t); // the hour for the given time
int hourFormat12(); // the hour now in 12 hour format
int hourFormat12(time_t t); // the hour for the given time in 12 hour format
uint8_t isAM(); // returns true if time now is AM
uint8_t isAM(time_t t); // returns true the given time is AM
uint8_t isPM(); // returns true if time now is PM
uint8_t isPM(time_t t); // returns true the given time is PM
int minute(); // the minute now
int minute(time_t t); // the minute for the given time
int second(); // the second now
int second(time_t t); // the second for the given time
int day(); // the day now
int day(time_t t); // the day for the given time
int weekday(); // the weekday now (Sunday is day 1)
int weekday(time_t t); // the weekday for the given time
int month(); // the month now (Jan is month 1)
int month(time_t t); // the month for the given time
int year(); // the full four digit year: (2009, 2010 etc)
int year(time_t t); // the year for the given time

time_t now(); // return the current time as seconds since Jan 1 1970
void setTime(time_t t);
void setTime(int hr,int min,int sec,int day, int month, int yr);
void adjustTime(long adjustment);

/* date strings */
#define dt_MAX_STRING_LEN 9 // length of longest date string (excluding terminating null)
char* monthStr(uint8_t month);
char* dayStr(uint8_t day);
char* monthShortStr(uint8_t month);
char* dayShortStr(uint8_t day);

/* time sync functions */
timeStatus_t timeStatus(); // indicates if time has been set and recently synchronized
void setSyncProvider( getExternalTime getTimeFunction); // identify the external time provider
void setSyncInterval(time_t interval); // set the number of seconds between re-sync

/* low level functions to convert to and from system time */
void breakTime(time_t time, tmElements_t &tm); // break time_t into elements
time_t makeTime(const tmElements_t &tm); // convert time elements into time_t

} // extern "C++"
#endif // __cplusplus
#endif /* _Time_h */
```

Source Code 1.12 Time.h

```
#include "TimeLib.h"
```

Source Code 1.13 TIme.cpp

```
/*
  time.c - low level time and date functions
  Copyright (c) Michael Margolis 2009-2014

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

  1.0 6 Jan 2010 - initial release
  1.1 12 Feb 2010 - fixed leap year calculation error
  1.2 1 Nov 2010 - fixed setTime bug (thanks to Korman for this)
  1.3 24 Mar 2012 - many edits by Paul Stoffregen: fixed timeStatus() to update
                    status, updated examples for Arduino 1.0, fixed ARM
                    compatibility issues, added TimeArduinoDue and TimeTeensy3
                    examples, add error checking and messages to RTC examples,
                    add examples to DS1307RTC library.
  1.4 5 Sep 2014 - compatibility with Arduino 1.5.7
*/

#if ARDUINO >= 100
#include <Arduino.h>
#else
#include <WProgram.h>
#endif

#include "TimeLib.h"

static tmElements_t tm; // a cache of time elements
static time_t cacheTime; // the time the cache was updated
static uint32_t syncInterval = 300; // time sync will be attempted after this many seconds

void refreshCache(time_t t) {
  if (t != cacheTime) {
    breakTime(t, tm);
    cacheTime = t;
  }
}

int hour() { // the hour now
  return hour(now());
}

```

```
int hour(time_t t) { // the hour for the given time
  refreshCache(t);
  return tm.Hour;
}

int hourFormat12() { // the hour now in 12 hour format
  return hourFormat12(now());
}

int hourFormat12(time_t t) { // the hour for the given time in 12 hour format
  refreshCache(t);
  if( tm.Hour == 0 )
    return 12; // 12 midnight
  else if( tm.Hour > 12)
    return tm.Hour - 12 ;
  else
    return tm.Hour ;
}

uint8_t isAM() { // returns true if time now is AM
  return !isPM(now());
}

uint8_t isAM(time_t t) { // returns true if given time is AM
  return !isPM(t);
}

uint8_t isPM() { // returns true if PM
  return isPM(now());
}

uint8_t isPM(time_t t) { // returns true if PM
  return (hour(t) >= 12);
}

int minute() {
  return minute(now());
}

int minute(time_t t) { // the minute for the given time
  refreshCache(t);
  return tm.Minute;
}

int second() {
  return second(now());
}

int second(time_t t) { // the second for the given time
  refreshCache(t);
  return tm.Second;
}

```

```
int day(){
  return(day(now()));
}

int day(time_t t) { // the day for the given time (0-6)
  refreshCache(t);
  return tm.Day;
}

int weekday() { // Sunday is day 1
  return weekday(now());
}

int weekday(time_t t) {
  refreshCache(t);
  return tm.Wday;
}

int month(){
  return month(now());
}

int month(time_t t) { // the month for the given time
  refreshCache(t);
  return tm.Month;
}

int year() { // as in Processing, the full four digit year: (2009, 2010 etc)
  return year(now());
}

int year(time_t t) { // the year for the given time
  refreshCache(t);
  return tmYearToCalendar(tm.Year);
}

/*==============================================================================*/
/* functions to convert to and from system time */
/* These are for interfacing with time serivces and are not normally needed in a sketch */

// leap year calulator expects year argument as years offset from 1970
#define LEAP_YEAR(Y) ( ((1970+(Y))>0) && !((1970+(Y))%4) && ( ((1970+(Y))%100) || !((1970+(Y))%400) ) )

static const uint8_t monthDays[]={31,28,31,30,31,30,31,31,30,31,30,31}; // API starts months from 1, this array starts from 0

void breakTime(time_t timeInput, tmElements_t &tm){
// break the given time_t into time components
// this is a more compact version of the C library localtime function
// note that year is offset from 1970 !!!
```

```

  uint8_t year;
  uint8_t month, monthLength;
  uint32_t time;
  unsigned long days;

  time = (uint32_t)timeInput;
  tm.Second = time % 60;
  time /= 60; // now it is minutes
  tm.Minute = time % 60;
  time /= 60; // now it is hours
  tm.Hour = time % 24;
  time /= 24; // now it is days
  tm.Wday = ((time + 4) % 7) + 1; // Sunday is day 1

  year = 0;
  days = 0;
  while((unsigned)(days += (LEAP_YEAR(year) ? 366 : 365)) <= time) {
    year++;
  }
  tm.Year = year; // year is offset from 1970

  days -= LEAP_YEAR(year) ? 366 : 365;
  time -= days; // now it is days in this year, starting at 0

  days=0;
  month=0;
  monthLength=0;
  for (month=0; month<12; month++) {
    if (month==1) { // february
      if (LEAP_YEAR(year)) {
        monthLength=29;
      } else {
        monthLength=28;
      }
    } else {
      monthLength = monthDays[month];
    }

    if (time >= monthLength) {
      time -= monthLength;
    } else {
        break;
    }
  }
  tm.Month = month + 1; // jan is month 1
  tm.Day = time + 1; // day of month
}

time_t makeTime(const tmElements_t &tm){
// assemble time elements into time_t
// note year argument is offset from 1970 (see macros in time.h to convert to other formats)
// previous version used full four digit year (or digits since 2000),i.e. 2009 was 2009 or 9
```

```

  int i;
  uint32_t seconds;

  // seconds from 1970 till 1 jan 00:00:00 of the given year
  seconds= tm.Year*(SECS_PER_DAY * 365);
  for (i = 0; i < tm.Year; i++) {
    if (LEAP_YEAR(i)) {
      seconds += SECS_PER_DAY; // add extra days for leap years
    }
  }

  // add days for this year, months start from 1
  for (i = 1; i < tm.Month; i++) {
    if ( (i == 2) && LEAP_YEAR(tm.Year)) {
      seconds += SECS_PER_DAY * 29;
    } else {
      seconds += SECS_PER_DAY * monthDays[i-1]; //monthDay array starts from 0
    }
  }
  seconds+= (tm.Day-1) * SECS_PER_DAY;
  seconds+= tm.Hour * SECS_PER_HOUR;
  seconds+= tm.Minute * SECS_PER_MIN;
  seconds+= tm.Second;
  return (time_t)seconds;
}
/*=====================================================*/
/* Low level system time functions */

static uint32_t sysTime = 0;
static uint32_t prevMillis = 0;
static uint32_t nextSyncTime = 0;
static timeStatus_t Status = timeNotSet;

getExternalTime getTimePtr; // pointer to external sync function
//setExternalTime setTimePtr; // not used in this version

#ifdef TIME_DRIFT_INFO // define this to get drift data
time_t sysUnsyncedTime = 0; // the time sysTime unadjusted by sync
#endif


time_t now() {
        // calculate number of seconds passed since last call to now()
  while (millis() - prevMillis >= 1000) {
                // millis() and prevMillis are both unsigned ints thus the subtraction will always be the
                    absolute value of the difference
    sysTime++;
    prevMillis += 1000;
#ifdef TIME_DRIFT_INFO
    sysUnsyncedTime++; // this can be compared to the synced time to measure long term drift
#endif
  }
  if (nextSyncTime <= sysTime) {
    if (getTimePtr != 0) {
      time_t t = getTimePtr();
      if (t != 0) {
        setTime(t);
      } else {
        nextSyncTime = sysTime + syncInterval;
        Status = (Status == timeNotSet) ? timeNotSet : timeNeedsSync;
      }
    }
  }
  return (time_t)sysTime;
}

void setTime(time_t t) {
#ifdef TIME_DRIFT_INFO
 if(sysUnsyncedTime == 0)
   sysUnsyncedTime = t; // store the time of the first call to set a valid Time
#endif

  sysTime = (uint32_t)t;
  nextSyncTime = (uint32_t)t + syncInterval;
  Status = timeSet;
  prevMillis = millis(); // restart counting from now (thanks to Korman for this fix)
}

void setTime(int hr,int min,int sec,int dy, int mnth, int yr){
 // year can be given as full four digit year or two digts (2010 or 10 for 2010);
 //it is converted to years since 1970
  if( yr > 99)
      yr = yr - 1970;
  else
      yr += 30;
  tm.Year = yr;
  tm.Month = mnth;
  tm.Day = dy;
  tm.Hour = hr;
  tm.Minute = min;
  tm.Second = sec;
  setTime(makeTime(tm));
}

void adjustTime(long adjustment) {
  sysTime += adjustment;
}

// indicates if time has been set and recently synchronized
timeStatus_t timeStatus() {
  now(); // required to actually update the status
  return Status;
}

void setSyncProvider( getExternalTime getTimeFunction){
  getTimePtr = getTimeFunction;
  nextSyncTime = sysTime;
  now(); // this will sync the clock
}

void setSyncInterval(time_t interval){ // set the number of seconds between re-sync
  syncInterval = (uint32_t)interval;
  nextSyncTime = sysTime + syncInterval;
}
```

Source Code 1.14 DateStrings.cpp

```
/* DateStrings.cpp
 * Definitions for date strings for use with the Time library
 *
 * Updated for Arduino 1.5.7 18 July 2014
 *
 * No memory is consumed in the sketch if your code does not call any of the string methods
 * You can change the text of the strings, make sure the short strings are each exactly 3 characters
 * the long strings can be any length up to the constant dt_MAX_STRING_LEN defined in TimeLib.h
 *
 */

#if defined(__AVR__)
#include <avr/pgmspace.h>
#else
// for compatiblity with Arduino Due and Teensy 3.0 and maybe others?
#define PROGMEM
#define PGM_P const char *
#define pgm_read_byte(addr) (*(const unsigned char *)(addr))
#define pgm_read_word(addr) (*(const unsigned char **)(addr))
#define strcpy_P(dest, src) strcpy((dest), (src))
#endif
#include <string.h> // for strcpy_P or strcpy
#include "TimeLib.h"

// the short strings for each day or month must be exactly dt_SHORT_STR_LEN
#define dt_SHORT_STR_LEN 3 // the length of short strings

static char buffer[dt_MAX_STRING_LEN+1]; // must be big enough for longest string and the
        terminating null

const char monthStr0[] PROGMEM = "";
const char monthStr1[] PROGMEM = "January";
const char monthStr2[] PROGMEM = "February";
const char monthStr3[] PROGMEM = "March";
const char monthStr4[] PROGMEM = "April";
const char monthStr5[] PROGMEM = "May";
const char monthStr6[] PROGMEM = "June";
const char monthStr7[] PROGMEM = "July";
const char monthStr8[] PROGMEM = "August";
const char monthStr9[] PROGMEM = "September";
const char monthStr10[] PROGMEM = "October";
const char monthStr11[] PROGMEM = "November";
const char monthStr12[] PROGMEM = "December";

const PROGMEM char * const PROGMEM monthNames_P[] =
{
    monthStr0,monthStr1,monthStr2,monthStr3,monthStr4,monthStr5,monthStr6,
    monthStr7,monthStr8,monthStr9,monthStr10,monthStr11,monthStr12
};

const char monthShortNames_P[] PROGMEM = "ErrJanFebMarAprMayJunJulAugSepOctNovDec";

const char dayStr0[] PROGMEM = "Err";
const char dayStr1[] PROGMEM = "Sunday";
const char dayStr2[] PROGMEM = "Monday";
const char dayStr3[] PROGMEM = "Tuesday";
const char dayStr4[] PROGMEM = "Wednesday";
const char dayStr5[] PROGMEM = "Thursday";
const char dayStr6[] PROGMEM = "Friday";
const char dayStr7[] PROGMEM = "Saturday";

const PROGMEM char * const PROGMEM dayNames_P[] =
{
   dayStr0,dayStr1,dayStr2,dayStr3,dayStr4,dayStr5,dayStr6,dayStr7
};

const char dayShortNames_P[] PROGMEM = "ErrSunMonTueWedThuFriSat";

/* functions to return date strings */

char* monthStr(uint8_t month)
{
    strcpy_P(buffer, (PGM_P)pgm_read_word(&(monthNames_P[month])));
    return buffer;
}

char* monthShortStr(uint8_t month)
{
   for (int i=0; i < dt_SHORT_STR_LEN; i++)
      buffer[i] = pgm_read_byte(&(monthShortNames_P[i+ (month*dt_SHORT_STR_LEN)]));
   buffer[dt_SHORT_STR_LEN] = 0;
   return buffer;
}

char* dayStr(uint8_t day)
{
   strcpy_P(buffer, (PGM_P)pgm_read_word(&(dayNames_P[day])));
   return buffer;
}

char* dayShortStr(uint8_t day)
{
   uint8_t index = day*dt_SHORT_STR_LEN;
   for (int i=0; i < dt_SHORT_STR_LEN; i++)
      buffer[i] = pgm_read_byte(&(dayShortNames_P[index + i]));
   buffer[dt_SHORT_STR_LEN] = 0;
   return buffer;
}
```