```
/**
 * 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, IPAddress poolServerIP) {
  this->_udp            = &udp;
  this->_poolServerIP   = poolServerIP;
  this->_poolServerName = NULL;
}

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

NTPClient::NTPClient(UDP& udp, IPAddress poolServerIP, long timeOffset){
  this->_udp            = &udp;
  this->_timeOffset     = timeOffset;
  this->_poolServerIP   = poolServerIP;
  this->_poolServerName = NULL;
}

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

NTPClient::NTPClient(UDP& udp, IPAddress poolServerIP, long timeOffset, unsigned
long updateInterval) {
  this->_udp            = &udp;
  this->_timeOffset     = timeOffset;
  this->_poolServerIP   = poolServerIP;
  this->_poolServerName = NULL;
  this->_updateInterval = updateInterval;
}

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

void NTPClient::begin(unsigned 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

  // flush any existing packets
  while(this->_udp->parsePacket() != 0)
    this->_udp->flush();

  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;  // return true after successful update
}

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->_port != NTP_DEFAULT_LOCAL_PORT)
this->begin(this->_port); // setup the UDP client if needed
    return this->forceUpdate();
  }
  return false;   // return false if update does not occur
}

bool NTPClient::isTimeSet() const {
  return (this->_lastUpdate != 0); // returns true if the time has been set,
else false
}

unsigned long NTPClient::getEpochTime() const {
  return this->_timeOffset + // User offset
         this->_currentEpoc + // Epoch 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
  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:
  if  (this->_poolServerName) {
    this->_udp->beginPacket(this->_poolServerName, 123);
  } else {
    this->_udp->beginPacket(this->_poolServerIP, 123);
  }
  this->_udp->write(this->_packetBuffer, NTP_PACKET_SIZE);
  this->_udp->endPacket();
}

void NTPClient::setRandomPort(unsigned int minValue, unsigned int maxValue) {
  randomSeed(analogRead(0));
  this->_port = random(minValue, maxValue);
}
```