```
/*************************************************************************
  This is an Arduino library for the Adafruit Thermal Printer.
  Pick one up at --> http://www.adafruit.com/products/597
  These printers use TTL serial to communicate, 2 pins are required.

  Adafruit invests time and resources providing this open source code.
  Please support Adafruit and open-source hardware by purchasing products
  from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution.
 *************************************************************************/

#if ARDUINO >= 100
 #include "Arduino.h"
#else
 #include "WProgram.h"
 #include "WConstants.h"
#endif
#include "Adafruit_Thermal.h"

// Though most of these printers are factory configured for 19200 baud
// operation, a few rare specimens instead work at 9600.  If so, change
// this constant.  This will NOT make printing slower!  The physical
// print and feed mechanisms are the limiting factor, not the port speed.
#define BAUDRATE  9600

// Number of microseconds to issue one byte to the printer.  11 bits
// (not 8) to accommodate idle, start and stop bits.  Idle time might
// be unnecessary, but erring on side of caution here.
#define BYTE_TIME (11L * 1000000L / BAUDRATE)

// Because there's no flow control between the printer and Arduino,
// special care must be taken to avoid overrunning the printer's buffer.
// Serial output is throttled based on serial speed as well as an estimate
// of the device's print and feed rates (relatively slow, being bound to
// moving parts and physical reality).  After an operation is issued to
// the printer (e.g. bitmap print), a timeout is set before which any
// other printer operations will be suspended.  This is generally more
// efficient than using delay() in that it allows the parent code to
// continue with other duties (e.g. receiving or decoding an image)
// while the printer physically completes the task.

// This method sets the estimated completion time for a just-issued task.
void Adafruit_Thermal::timeoutSet(unsigned long x) {
  resumeTime = micros() + x;
}

// This function waits (if necessary) for the prior task to complete.
void Adafruit_Thermal::timeoutWait() {
  while((long)(micros() - resumeTime) < 0L); // Rollover-proof
}

// Printer performance may vary based on the power supply voltage,
// thickness of paper, phase of the moon and other seemingly random
// variables.  This method sets the times (in microseconds) for the
// paper to advance one vertical 'dot' when printing and feeding.
```

```
// For example, in the default initialized state, normal-sized text is
// 24 dots tall and the line spacing is 32 dots, so the time for one
// line to be issued is approximately 24 * print time + 8 * feed time.
// The default print and feed times are based on a random test unit,
// but as stated above your reality may be influenced by many factors.
// This lets you tweak the timing to avoid excessive delays and/or
// overrunning the printer buffer.
void Adafruit_Thermal::setTimes(unsigned long p, unsigned long f) {
  dotPrintTime = p;
  dotFeedTime  = f;
}

// Constructor
Adafruit_Thermal::Adafruit_Thermal(int RX_Pin, int TX_Pin) {
  _RX_Pin = RX_Pin;
  _TX_Pin = TX_Pin;
}

// The next four helper methods are used when issuing configuration
// commands, printing bitmaps or barcodes, etc.  Not when printing text.

void Adafruit_Thermal::writeBytes(uint8_t a) {
  timeoutWait();
  PRINTER_PRINT(a);
  timeoutSet(BYTE_TIME);
}

void Adafruit_Thermal::writeBytes(uint8_t a, uint8_t b) {
  timeoutWait();
  PRINTER_PRINT(a);
  PRINTER_PRINT(b);
  timeoutSet(2 * BYTE_TIME);
}

void Adafruit_Thermal::writeBytes(uint8_t a, uint8_t b, uint8_t c) {
  timeoutWait();
  PRINTER_PRINT(a);
  PRINTER_PRINT(b);
  PRINTER_PRINT(c);
  timeoutSet(3 * BYTE_TIME);
}

void Adafruit_Thermal::writeBytes(uint8_t a, uint8_t b, uint8_t c, uint8_t d) {
  timeoutWait();
  PRINTER_PRINT(a);
  PRINTER_PRINT(b);
  PRINTER_PRINT(c);
  PRINTER_PRINT(d);
  timeoutSet(3 * BYTE_TIME);
}

// The underlying method for all high-level printing (e.g. println()).
// The inherited Print class handles the rest!
#if ARDUINO >= 100
size_t Adafruit_Thermal::write(uint8_t c) {
#else
void Adafruit_Thermal::write(uint8_t c) {
```

```
#endif

  if(c != 0x13) { // Strip carriage returns
    timeoutWait();
    PRINTER_PRINT(c);
    unsigned long d = BYTE_TIME;
    if((c == '\n') || (column == maxColumn)) { // If newline or wrap
      d += (prevByte == '\n') ?
        ((charHeight+lineSpacing) * dotFeedTime) :             // Feed line
        ((charHeight*dotPrintTime)+(lineSpacing*dotFeedTime)); // Text line
      column = 0;
      c      = '\n'; // Treat wrap as newline on next pass
    } else {
      column++;
    }
    timeoutSet(d);
    prevByte = c;
  }

#if ARDUINO >= 100
  return 1;
#endif
}

void Adafruit_Thermal::begin(int heatTime) {
  _printer = new SERIAL_IMPL(_RX_Pin, _TX_Pin);
  _printer->begin(BAUDRATE);

  // The printer can't start receiving data immediately upon power up --
  // it needs a moment to cold boot and initialize.  Allow at least 1/2
  // sec of uptime before printer can receive data.
  timeoutSet(500000L);

  wake();
  reset();

  // Description of print settings from page 23 of the manual:
  // ESC 7 n1 n2 n3 Setting Control Parameter Command
  // Decimal: 27 55 n1 n2 n3
  // Set "max heating dots", "heating time", "heating interval"
  // n1 = 0-255 Max printing dots, Unit (8dots), Default: 7 (64 dots)
  // n2 = 3-255 Heating time, Unit (10us), Default: 80 (800us)
  // n3 = 0-255 Heating interval, Unit (10us), Default: 2 (20us)
  // The more max heating dots, the more peak current will cost
  // when printing, the faster printing speed. The max heating
  // dots is 8*(n1+1).  The more heating time, the more density,
  // but the slower printing speed.  If heating time is too short,
  // blank page may occur.  The more heating interval, the more
  // clear, but the slower printing speed.

  writeBytes(27, 55);   // Esc 7 (print settings)
  writeBytes(20);       // Heating dots (20=balance of darkness vs no jams)
  writeBytes(heatTime); // Library default = 255 (max)
  writeBytes(250);      // Heat interval (500 uS = slower, but darker)

  // Description of print density from page 23 of the manual:
  // DC2 # n Set printing density
```

```
  // Decimal: 18 35 n
  // D4..D0 of n is used to set the printing density.  Density is
  // 50% + 5% * n(D4-D0) printing density.
  // D7..D5 of n is used to set the printing break time.  Break time
  // is n(D7-D5)*250us.
  // (Unsure of the default value for either -- not documented)

#define printDensity   14 // 120% (? can go higher, text is darker but fuzzy)
#define printBreakTime  4 // 500 uS

  writeBytes(18, 35); // DC2 # (print density)
  writeBytes((printBreakTime << 5) | printDensity);

  dotPrintTime = 30000; // See comments near top of file for
  dotFeedTime  =  2100; // an explanation of these values.
}

// Reset printer to default state.
void Adafruit_Thermal::reset() {
  prevByte      = '\n'; // Treat as if prior line is blank
  column        = 0;
  maxColumn     = 32;
  charHeight    = 24;
  lineSpacing   = 8;
  barcodeHeight = 50;
  writeBytes(27, 64);
}

// Reset text formatting parameters.
void Adafruit_Thermal::setDefault(){
  online();
  justify('L');
  inverseOff();
  doubleHeightOff();
  setLineHeight(32);
  boldOff();
  underlineOff();
  setBarcodeHeight(50);
  setSize('s');
}

void Adafruit_Thermal::test(){
  println("Hello World!");
  feed(2);
}

void Adafruit_Thermal::testPage() {
  writeBytes(18, 84);
  timeoutSet(
    dotPrintTime * 24 * 26 +      // 26 lines w/text (ea. 24 dots high)
    dotFeedTime * (8 * 26 + 32)); // 26 text lines (feed 8 dots) + blank line
}

void Adafruit_Thermal::setBarcodeHeight(int val) { // Default is 50
  if(val < 1) val = 1;
  barcodeHeight = val;
  writeBytes(29, 104, val);
```

```
}

void Adafruit_Thermal::printBarcode(char * text, uint8_t type) {
  int  i = 0;
  byte c;

  writeBytes(29,  72, 2);    // Print label below barcode
  writeBytes(29, 119, 3);    // Barcode width
  writeBytes(29, 107, type); // Barcode type (listed in .h file)
  do { // Copy string + NUL terminator
    writeBytes(c = text[i++]);
  } while(c);
  timeoutSet((barcodeHeight + 40) * dotPrintTime);
  prevByte = '\n';
  feed(2);
}

// === Character commands ===

#define INVERSE_MASK       (1 << 1)
#define UPDOWN_MASK        (1 << 2)
#define BOLD_MASK          (1 << 3)
#define DOUBLE_HEIGHT_MASK (1 << 4)
#define DOUBLE_WIDTH_MASK  (1 << 5)
#define STRIKE_MASK        (1 << 6)

void Adafruit_Thermal::setPrintMode(uint8_t mask) {
  printMode |= mask;
  writePrintMode();
  charHeight = (printMode & DOUBLE_HEIGHT_MASK) ? 48 : 24;
  maxColumn  = (printMode & DOUBLE_WIDTH_MASK ) ? 16 : 32;
}
void Adafruit_Thermal::unsetPrintMode(uint8_t mask) {
  printMode &= ~mask;
  writePrintMode();
  charHeight = (printMode & DOUBLE_HEIGHT_MASK) ? 48 : 24;
  maxColumn  = (printMode & DOUBLE_WIDTH_MASK ) ? 16 : 32;
}

void Adafruit_Thermal::writePrintMode() {
  writeBytes(27, 33, printMode);
}

void Adafruit_Thermal::normal() {
  printMode = 0;
  writePrintMode();
}

void Adafruit_Thermal::inverseOn(){
  setPrintMode(INVERSE_MASK);
}

void Adafruit_Thermal::inverseOff(){
  unsetPrintMode(INVERSE_MASK);
}

void Adafruit_Thermal::upsideDownOn(){
```

```
  setPrintMode(UPDOWN_MASK);
}

void Adafruit_Thermal::upsideDownOff(){
  unsetPrintMode(UPDOWN_MASK);
}

void Adafruit_Thermal::doubleHeightOn(){
  setPrintMode(DOUBLE_HEIGHT_MASK);
}

void Adafruit_Thermal::doubleHeightOff(){
  unsetPrintMode(DOUBLE_HEIGHT_MASK);
}

void Adafruit_Thermal::doubleWidthOn(){
  setPrintMode(DOUBLE_WIDTH_MASK);
}

void Adafruit_Thermal::doubleWidthOff(){
  unsetPrintMode(DOUBLE_WIDTH_MASK);
}

void Adafruit_Thermal::strikeOn(){
  setPrintMode(STRIKE_MASK);
}

void Adafruit_Thermal::strikeOff(){
  unsetPrintMode(STRIKE_MASK);
}

void Adafruit_Thermal::boldOn(){
  setPrintMode(BOLD_MASK);
}

void Adafruit_Thermal::boldOff(){
  unsetPrintMode(BOLD_MASK);
}

void Adafruit_Thermal::justify(char value){
  uint8_t pos = 0;

  switch(toupper(value)) {
    case 'L': pos = 0; break;
    case 'C': pos = 1; break;
    case 'R': pos = 2; break;
  }

  writeBytes(0x1B, 0x61, pos);
}

// Feeds by the specified number of lines
void Adafruit_Thermal::feed(uint8_t x){
  // The datasheet claims sending bytes 27, 100, <x> will work, but
  // it feeds much more than that.  So it's done manually:
  while(x--) write('\n');
}
```

```
// Feeds by the specified number of individual  pixel rows
void Adafruit_Thermal::feedRows(uint8_t rows) {
  writeBytes(27, 74, rows);
  timeoutSet(rows * dotFeedTime);
}

void Adafruit_Thermal::flush() {
  writeBytes(12);
}

void Adafruit_Thermal::setSize(char value){
  uint8_t size;

  switch(toupper(value)) {
   default:  // Small: standard width and height
    size       = 0x00;
    charHeight = 24;
    maxColumn  = 32;
    break;
   case 'M': // Medium: double height
    size       = 0x01;
    charHeight = 48;
    maxColumn  = 32;
    break;
   case 'L': // Large: double width and height
    size       = 0x11;
    charHeight = 48;
    maxColumn  = 16;
    break;
  }

  writeBytes(29, 33, size, 10);
  prevByte = '\n'; // Setting the size adds a linefeed
}

// Underlines of different weights can be produced:
// 0 - no underline
// 1 - normal underline
// 2 - thick underline
void Adafruit_Thermal::underlineOn(uint8_t weight) {
  writeBytes(27, 45, weight);
}

void Adafruit_Thermal::underlineOff() {
  underlineOn(0);
}

void Adafruit_Thermal::printBitmap(
 int w, int h, const uint8_t *bitmap, bool fromProgMem) {
  int rowBytes, rowBytesClipped, rowStart, chunkHeight, x, y, i;

  rowBytes        = (w + 7) / 8; // Round up to next byte boundary
  rowBytesClipped = (rowBytes >= 48) ? 48 : rowBytes; // 384 pixels max width

  for(i=rowStart=0; rowStart < h; rowStart += 255) {
    // Issue up to 255 rows at a time:
```

```
    chunkHeight = h - rowStart;
    if(chunkHeight > 255) chunkHeight = 255;

    writeBytes(18, 42, chunkHeight, rowBytesClipped);

    for(y=0; y < chunkHeight; y++) {
      for(x=0; x < rowBytesClipped; x++, i++) {
        PRINTER_PRINT(fromProgMem ? pgm_read_byte(bitmap + i) : *(bitmap+i));
      }
      i += rowBytes - rowBytesClipped;
    }
    timeoutSet(chunkHeight * dotPrintTime);
  }
  prevByte = '\n';
}

void Adafruit_Thermal::printBitmap(int w, int h, Stream *stream) {
  int rowBytes, rowBytesClipped, rowStart, chunkHeight, x, y, i, c;

  rowBytes        = (w + 7) / 8; // Round up to next byte boundary
  rowBytesClipped = (rowBytes >= 48) ? 48 : rowBytes; // 384 pixels max width

  for(rowStart=0; rowStart < h; rowStart += 255) {
    // Issue up to 255 rows at a time:
    chunkHeight = h - rowStart;
    if(chunkHeight > 255) chunkHeight = 255;

    writeBytes(18, 42, chunkHeight, rowBytesClipped);

    for(y=0; y < chunkHeight; y++) {
      for(x=0; x < rowBytesClipped; x++) {
        while((c = stream->read()) < 0);
        PRINTER_PRINT((uint8_t)c);
      }
      for(i = rowBytes - rowBytesClipped; i>0; i--) {
        while((c = stream->read()) < 0);
      }
    }
    timeoutSet(chunkHeight * dotPrintTime);
  }
  prevByte = '\n';
}

void Adafruit_Thermal::printBitmap(Stream *stream) {
  uint8_t  tmp;
  uint16_t width, height;

  tmp    =  stream->read();
  width  = (stream->read() << 8) + tmp;

  tmp    =  stream->read();
  height = (stream->read() << 8) + tmp;

  printBitmap(width, height, stream);
}

// Take the printer offline. Print commands sent after this will be
```

```
// ignored until 'online' is called.
void Adafruit_Thermal::offline(){
  writeBytes(27, 61, 0);
}

// Take the printer back online. Subsequent print commands will be obeyed.
void Adafruit_Thermal::online(){
  writeBytes(27, 61, 1);
}

// Put the printer into a low-energy state immediately.
void Adafruit_Thermal::sleep() {
  sleepAfter(1);
}

// Put the printer into a low-energy state after the given number
// of seconds.
void Adafruit_Thermal::sleepAfter(uint8_t seconds) {
  writeBytes(27, 56, seconds);
}

// Wake the printer from a low-energy state.
void Adafruit_Thermal::wake() {
  // Printer may have been idle for a very long time, during which the
  // micros() counter has rolled over.  To avoid shenanigans, reset the
  // timeout counter before issuing the wake command.
  timeoutSet(0);
  writeBytes(255);
  // Datasheet recomments a 50 mS delay before issuing further commands,
  // but in practice this alone isn't sufficient (e.g. text size/style
  // commands may still be misinterpreted on wake).  A slightly longer
  // delay, interspersed with ESC chars (no-ops) seems to help.
  for(uint8_t i=0; i<10; i++) {
    writeBytes(27);
    timeoutSet(10000L);
  }
}

// Tell the soft serial to listen. Needed if you are using multiple
// SoftSerial interfaces.
void Adafruit_Thermal::listen() {
  _printer->listen();
}

// Check the status of the paper using the printers self reporting
// ability. Doesn't match the datasheet...
// Returns true for paper, false for no paper.
bool Adafruit_Thermal::hasPaper() {
  writeBytes(27, 118, 0);

  char stat;
  // Some delay while checking.
  // Could probably be done better...
  for (int i = 0; i < 1000; i++) {
    if (_printer->available()) {
      stat = _printer->read();
      break;
```

```
    }
  }

  // Mask the 3 LSB, this seems to be the one we care about.
  stat = stat & 0b000100;

  // If it's set, no paper, if it's clear, we have paper.
  if (stat == 0b000100) {
    return false;
  } else if (stat == 0b000000){
    return true;

  }

}

void Adafruit_Thermal::setLineHeight(int val) {
  if(val < 24) val = 24;
  lineSpacing = val - 24;

  // The printer doesn't take into account the current text height
  // when setting line height, making this more akin to inter-line
  // spacing.  Default line spacing is 32 (char height of 24, line
  // spacing of 8).
  writeBytes(27, 51, val);
}

////////////////////// not working?
void Adafruit_Thermal::tab() {
  PRINTER_PRINT(9);
}
void Adafruit_Thermal::setCharSpacing(int spacing) {
  writeBytes(27, 32, 0, 10);
}
//////////////////////////

#if ARDUINO < 100
void *operator new(size_t size_) { return malloc(size_); }
void* operator new(size_t size_,void *ptr_) { return ptr_; }
#endif
```