Cette étude porte sur l'afficheur de **Pimoroni** « **Display-O-Tron HAT »** qui dispose de trois lignes d'affichage de 16 caractères. Cet afficheur a des dimensions qui correspondent à celles de la « **Raspberry** » et se branche directement sur le connecteur de cette dernière. La particularité de cet afficheur, c'est qu'il possède un retro-éclairage dont nous pouvons choisir la couleur avec éventuellement des valeurs différentes sur toute la largeur.

> ***Nos travaux pourront s'effectuer directement sur la « Raspberry » en mode console ou sur l'ordinateur de développement avec l'éditeur que nous utilisons habituellement en les déployant ensuite à l'aide de la commande « scp ».***
>
> ***À la fin de notre étude, nous réaliserons également un service permettant de gérer l'afficheur à distance.***

INSTALLATION DE TOUS LES OUTILS NÉCESSAIRES AU BON FONCTIONNEMENT DE CET AFFICHEUR

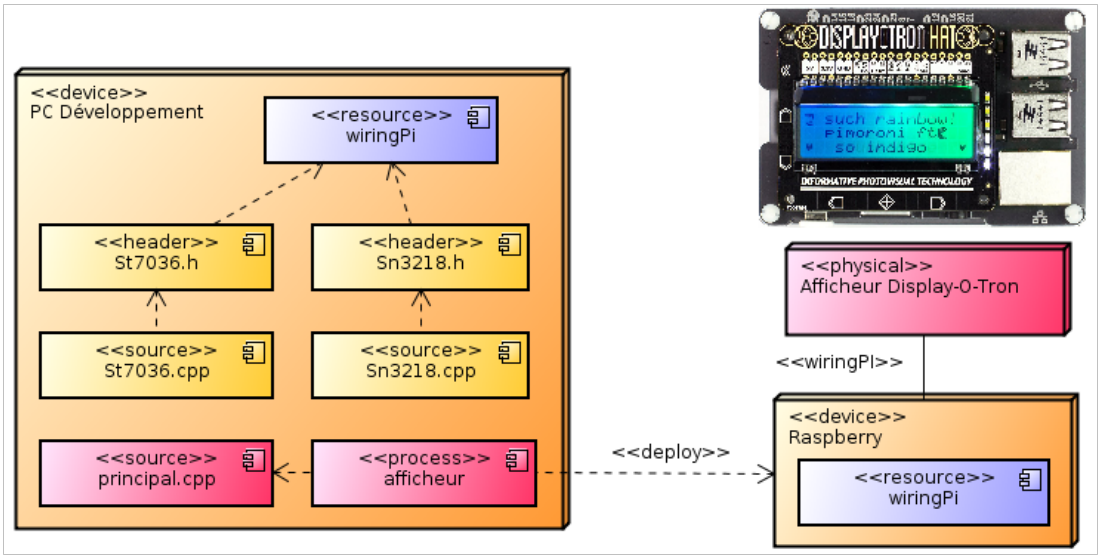
Afin de pouvoir exploiter pleinement des compétences de cet afficheur, nous devons réaliser l'installation des outils et des sources (scripts) nécessaires pour réaliser nos programmes le plus simplement possible. Cette installation se fera directement sur la « **Raspberry** » :

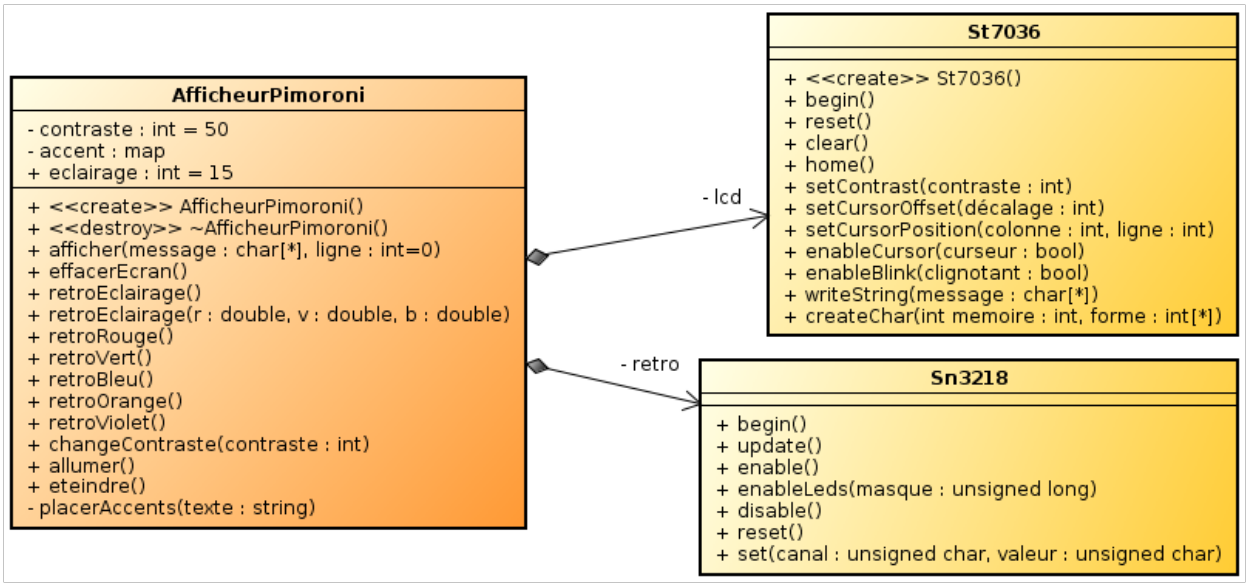
> ***https://learn.pimoroni.com/tutorial/display-o-tron/getting-started-with-display-o-tron*** *// Lien explicatif*
> ***$ curl https://get.pimoroni.com/displayotron | bash*** *// Téléchargement et installation complète (répondre « **yes** » deux fois).*
>
> ***Cette installation est indispensable si nous souhaitons réaliser un développement avec le langage python. Dans le cadre du langage C++, nous devons intégrer la bibliothèque qui permet de gérer tous les composants intégrés à la Raspberry avec un accès direct. Cette bibliothèque s'appelle « wiringPi » . Si vous souhaitez faire de la compilation croisée, cette bibliothèque doit être également intégrée au poste de développement distant.***
>
> ***$ sudo dpkg -i wiringpi_2.44_armhf.deb***

DÉVELOPPEMENT SUR L'ORDINATEUR DE DÉVELOPPEMENT

Lorsque nous réalisons des développement à distance en dehors de la « Raspberry », sur l'ordinateur de développement, nous devons disposer des classes relatives au fonctionnement complet de l'afficheur « **Display-O-HAT** ». Il en existe deux classes importantes qui correspondent chacune à une gestion spécifique de l'afficheur :

1. ***St7036*** : qui s'occupe de l'affichage des différents messages sur les trois lignes disponibles avec la possibilité d'afficher ou pas un curseur clignotant.
2. ***Sn3218*** : qui s'occupe du rétro éclairage avec toutes les couleurs possibles ainsi que d'un « **bargraph** » sur le côté droit de l'afficheur.





MÉTHODES ASSOCIÉES À L'OBJET « LCD » DÉFINI DANS LA CLASSE ST7036

méthode	Description de la méthode
begin()	*Avant toute manipulation avec l'afficheur, vous devez impérativement initialiser les commandes associées à l'affichage de texte grâce à cette méthode. Sans cela, les méthodes suivantes ne fonctionnerons pas.*
reset()	*Réinitialise l'afficheur.*
clear()	*Efface complètement tout le contenu actuel de l'afficheur et place le curseur d'affichage au début de la première ligne.*
home()	*Place le curseur d'affichage sur la première ligne et sur la première colonne.*
setContrast(contraste)	*Permet de définir une valeur de contraste comprise entre **0** et **63**. Cette valeur agit uniquement sur le texte et non pas sur le rétro éclairage. Une valeur aux alentours de **50** permet de bien discerner le texte même si notre regard n'est pas tout à fait dans l'axe de vision.*
setCursorPosition(colonne, ligne)	*Déplace le curseur d'affichage suivant une valeur de colonne de **0** à **15** et un choix de ligne de **0** à **2**.*
setCursorOffset(décalage)	*Place le curseur d'affichage suivant un décalage par rapport à la position d'origine sachant que les lignes sont consécutives en mémoire (c'est comme s'il n'y avait qu'une seule ligne d'affichage de **3**x**16** caractères).*
enableCursor(validation)	*Spécifie avec une valeur booléenne (**true** ou **false**) l'affichage du curseur. Par défaut, il n'est pas visible.*
enableBlink(validation)	*Spécifie par une valeur booléenne (**true** ou **false**) le clignotement du curseur. Par défaut, il n'est pas clignotant.*
writeString(message)	*Demande d'afficher un texte à l'endroit où se situe le curseur d'affichage. Si la longueur du texte dépasse la ligne actuelle, l'affichage se poursuit automatiquement sur la ligne suivante.*
create_char(mémoire, forme)	*Permet de créer un nouveau caractère personnalisé sauvegardé en interne dans la mémoire de l'afficheur. Il est possible d'en stocker **8** au maximum, « **mémoire** » correspond au numéro de stockage de **0** à **7**, **« forme »** est un tableau de **8** entiers de 6 bits chacun.*

Voici ci-dessous les codes sources complets relatif à cette classe :

St7036.h

```
#ifndef ST7036_H
#define ST7036_H
#include <stdint.h>

class St7036 {
  public:
        St7036(int register_select_pin = 6, int reset_pin = 26, int rows = 3, int columns = 16,
               int spi_chip_select=0);
        void begin();
        void reset();
        void clear();
        void home();
        void setContrast(int contrast);
        void setCursorOffset(int offset);
        void setCursorPosition(int column, int row);
        void enableCursor(bool cursor);
        void enableBlink(bool blink);
        void writeString(const char* str, int length = 0);
        void createChar(int charPos, int charMap[]);
  private:
        int _resetPin;
        int _regPin;
        int _rows;
        int _columns;
        int _cs;
        int _fdSpi;
        bool _enabled = 1;
        bool _cursorEnabled = 0;
        bool _cursorBlink = 0;
        bool _doubleHeight = 0;
        void writeByte(uint8_t b);
        void writeInstructionSet(int instruction_set = 0);
        void writeCommand(uint8_t value, int instruction_set = 0);
        void updateDisplayMode();
        void setBias(int bias=1);
        void writeChar(int value);
        static const int INSTRUCTION_SET_TEMPLATE = 0b00111000;
        // commands
        static const int COMMAND_CLEAR  = 0b00000001;
        static const int COMMAND_HOME   = 0b00000010;
        static const int COMMAND_SCROLL = 0b00010000;
        static const int COMMAND_DOUBLE = 0b00010000;
        static const int COMMAND_BIAS   = 0b00010100;
        static const int COMMAND_SET_DISPLAY_MODE = 0b00001000;
```

```
      // display modes
       static const int BLINK_ON   = 0b00000001;
       static const int CURSOR_ON  = 0b00000010;
       static const int DISPLAY_ON = 0b00000100;
};

#endif
```

St7036.cpp

```
#include "St7036.h"
#include <cstring>
#include <unistd.h> // read()/write()
#include <wiringPi.h>
#include <wiringPiSPI.h>

St7036::St7036(int register_select_pin, int reset_pin, int rows, int columns, int spi_chip_select) {
        _resetPin = reset_pin;
        _regPin = register_select_pin;
        _rows = rows;
        _columns = columns;
        _cs = spi_chip_select;
}

void St7036::begin() {
        _fdSpi = wiringPiSPISetup(_cs, 1000000);
        pinMode(_resetPin, OUTPUT);
        pinMode(_regPin, OUTPUT);
        reset();
        digitalWrite(_regPin, HIGH);
        updateDisplayMode();
        // set entry mode (no shift, cursor direction)
        writeCommand(0b00000100 | 0b00000010);
        // ???
        setBias(1);
        setContrast(40);
        clear();
}

void St7036::reset() {
        digitalWrite(_resetPin, LOW);
        delay(1);
        digitalWrite(_resetPin, HIGH);
        delay(1);
}

void St7036::writeInstructionSet(int instruction_set) {
        digitalWrite(_regPin, LOW);
        writeByte(INSTRUCTION_SET_TEMPLATE | instruction_set | (_doubleHeight << 2));
        delay(1);
}

void St7036::writeByte(uint8_t b) {
        uint8_t buf[1];
        buf[0] = b;
        //wiringPiSPIDataRW(_cs, buf, 1);
        write(_fdSpi, buf, 1);
}

void St7036::writeCommand(uint8_t value, int instruction_set) {
        digitalWrite(_regPin, LOW);
        writeInstructionSet(instruction_set);
        writeByte(value);
        delay(1);
}

void St7036::updateDisplayMode() {
        int mask = COMMAND_SET_DISPLAY_MODE;
        mask |= DISPLAY_ON * _enabled;
        mask |= CURSOR_ON * _cursorEnabled;
        mask |= BLINK_ON * _cursorBlink;
        writeCommand(mask);
}

void St7036::setCursorOffset(int offset) {
        writeCommand(0b10000000 | offset);
}

void St7036::setCursorPosition(int column, int row) {
        int offset = _columns * row + column;
        writeCommand(0b10000000 | offset);
}
```

```
void St7036::home() {
        setCursorPosition(0, 0);
}

void St7036::clear() {
        writeCommand(COMMAND_CLEAR);
        home();
}

void St7036::setBias(int bias) {
        writeCommand(COMMAND_BIAS | (bias << 4) | 1, 1);
}

void St7036::setContrast(int contrast) {
        if (contrast < 0 || contrast > 0x3F) {
                return;
        }
        // For 3.3v operation the booster must be on, which is
        // on the same command as the (2-bit) high-nibble of contrast
        writeCommand((0b01010100 | ((contrast >> 4) & 0x03)), 1);
        writeCommand(0b01101011, 1);
        // Set low-nibble of the contrast
        writeCommand((0b01110000 | (contrast & 0x0F)), 1);
}

void St7036::enableCursor(bool cursor) {
        _cursorEnabled = cursor;
        updateDisplayMode();
}

void St7036::enableBlink(bool blink) {
        _cursorBlink = blink;
        updateDisplayMode();
}

void St7036::writeString(const char* str, int length) {
        digitalWrite(_regPin, HIGH);
        length = length <= 0? strlen(str) : length;
        for (int i = 0; i < length; i++){
                writeByte(str[i]);
        }
        delay(1);
}

void St7036::writeChar(int value) {
        digitalWrite(_regPin, HIGH);
        writeByte(value);
}

void St7036::createChar(int charPos, int charMap[]) {
        if (charPos < 0 || charPos > 7) {
                return;
        }
        int baseAddress = charPos*8;
        for (int i = 0; i < 8; i++) {
                writeCommand(0x40 | (baseAddress+i));
                writeChar(charMap[i]);
        }
}
```

MÉTHODES ASSOCIÉES À L'OBJET «RETRO» DÉFINI DANS LA CLASSE SN3218

Sur la partie haute de l'afficheur, vous disposez de 6 zones de rétro-éclairage dont chacune est composée de trois LED correspondant aux trois couleurs fondamentales, dans l'ordre le bleu, le vert et le rouge.

méthode	Description de la méthode
begin()	*Avant toute manipulation avec l'afficheur, vous devez impérativement initialiser les commandes associées à la gestion du rétro-éclairage grâce à cette méthode. Sans cela, les méthodes suivantes ne fonctionnerons pas.*
reset()	*Réinitialise le rétro-éclairage.*
update()	*Permet de prendre en compte les dernières modifications proposées pour le rétro-éclairage.*
disable()	*Désactive le rétro-éclairage.*
enable()	*Réactive le rétro-éclairage avec les réglages proposées précédemment.*
enableLeds(masque)	*Sur la partie haute de l'afficheur, vous disposez de **6** zones de rétro-éclairage dont chacune est composée de trois LEDs correspondant aux trois couleurs fondamentales, dans l'ordre le bleu, le vert et le rouge. Cette classe dispose également de constantes publiques et statiques pour le choix des LEDs à traiter : **CH_00** à **CH_17** et si nous choisissons toutes les LEDs : **CH_ALL**. Vous devez spécifier le **masque** (les LEDs à prendre en compte) à l'aide de ces constantes.*

set(canal, valeur)	*Une fois que le choix des LEDs est effectué avec la méthode précédente, vous spécifiez l'intensité de la **valeur** pour chacune des LEDs choisies (**canal**). Si vous prenez tout le rétro-éclairage, vous devez régler tous les canaux.*

Voici ci-dessous les codes sources complets relatif à cette classe :

Sn3218.h

```
#ifndef SN3218_H
#define SN3218_H
#include <stdint.h>

class Sn3218 {
  public:
        void begin();
        void update();
        void enable();
        void enableLeds( unsigned long enable_mask );
        void disable();
        void reset();
        void set( unsigned char chan, unsigned char val );
        static const int NUM_CHANNELS = 18;
        // use with enableLeds()
        static const int CH_ALL = 0x3FFFF;
        static const int CH_00 = 0x00001;
        static const int CH_01 = 0x00002;
        static const int CH_02 = 0x00004;
        static const int CH_03 = 0x00008;
        static const int CH_04 = 0x00010;
        static const int CH_05 = 0x00020;
        static const int CH_06 = 0x00040;
        static const int CH_07 = 0x00080;
        static const int CH_08 = 0x00100;
        static const int CH_09 = 0x00200;
        static const int CH_10 = 0x00400;
        static const int CH_11 = 0x00800;
        static const int CH_12 = 0x01000;
        static const int CH_13 = 0x02000;
        static const int CH_14 = 0x04000;
        static const int CH_15 = 0x08000;
        static const int CH_16 = 0x10000;
        static const int CH_17 = 0x20000;
  private:
        int _fdDev;
        void writeReg( unsigned char reg, unsigned char val );
        static const int DEV_ADDR = 0x54;
        static const int CMD_ENABLE_OUTPUT = 0x00;
        static const int CMD_SET_PWM_VALUES = 0x01;
        static const int CMD_ENABLE_LEDS = 0x13;
        static const int CMD_UPDATE = 0x16;
        static const int CMD_RESET = 0x17;
};
#endif
```

Sn3218.cpp

```
#include "Sn3218.h"
#include <unistd.h> // write()
#include <wiringPiI2C.h>


void Sn3218::begin() {
        _fdDev = wiringPiI2CSetup(DEV_ADDR);
        enable();
}

void Sn3218::update() {
        this->writeReg(CMD_UPDATE, 0x00);
}

void Sn3218::reset() {
        writeReg(CMD_RESET, 0x00);
        enable();
}
void Sn3218::enable() {
        writeReg(CMD_ENABLE_OUTPUT, 0x01);
}

void Sn3218::disable() {
        writeReg(CMD_ENABLE_OUTPUT, 0x00);
}
```

```
void Sn3218::enableLeds(unsigned long enable_mask) {
        writeReg(CMD_ENABLE_LEDS, enable_mask & 0x3F);
        writeReg(CMD_ENABLE_LEDS + 1, (enable_mask >> 6) & 0x3F);
        writeReg(CMD_ENABLE_LEDS + 2, (enable_mask >> 12) & 0x3F);
        update();
}

void Sn3218::set(unsigned char chan, unsigned char val) {
        writeReg(CMD_SET_PWM_VALUES + chan, val);
}

void Sn3218::writeReg(unsigned char reg, unsigned char val) {
        uint8_t buffer[2];
        buffer[0] = reg;
        buffer[1] = val;
        write(_fdDev, buffer, 2);
}
```

CRÉATION DE LA CLASSE AFFICHEURPIMORONI

Afin de factoriser l'ensemble des deux classes qui gèrent cet afficheur, pour l'affichage de messages à l'endroit souhaité et aussi pour une gestion correcte du rétro-éclairage, je vous propose de fabriquer une classe qui permet de simplifier considérablement l'utilisation de cet afficheur.

Notamment, lorsque nous créons un objet de cette classe, il faut qu'il soit prêt à être exploité avec l'activation des deux parties, avoir un bon réglage d'éclairage et de contraste dès le départ et de désactiver le rétro-éclairage lorsque l'objet est détruit.

*Il faut également que cette classe fasse en sorte d'afficher n'importe quel message avec des accents. Or, le texte compilé en C++ prend en compte le codage **UTF-8**, ce qui n'est pas le cas de l'afficheur qui lui n'utilise que le code **ASCII étendu** de type **OEM**. Nous avons donc une méthode privée qui réalise automatiquement cette conversion.*

attribut	Description de l'attribut
contraste	*Nous définissons un contraste par défaut qui correspond me semble-t-il au meilleur choix pour avoir une meilleure visibilité du texte affiché.*
eclairage	*Nous prévoyons également un réglage qui correspond à la meilleure intensité du rétro-éclairage, pour que cela soit bien visible et sans agressivité.*
accent	*Dictionnaire de décryptage qui comporte l'ensemble des accents à prendre en compte dans la langue française.*

méthode	Description de la méthode
AfficheurPimoroni()	*Constructeur qui réalise l'initialisation complète de l'afficheur avec l'activation du rétro-éclairage.*
~AfficheurPimoroni()	*Destructeur qui éteint le rétro-éclairage.*
placerAccents(texte)	*Méthode privée qui retourne un texte exploitable par l'afficheur pour une bonne gestion des accents utilisés par la langue française.*
afficher(message, ligne)	*Permet d'afficher le texte souhaité en choisissant éventuellement la ligne de l'afficheur. Par défaut, il s'agit de la première ligne.*
effacerEcran()	*Enlève tout le texte présent actuellement sur l'afficheur.*
retroEclairage()	*Prévoit un rétro-éclairage complet par défaut en prenant en compte la luminosité présélectionnée.*
retroEclairage(rouge, vert, bleu)	*Prévoit un rétro-éclairage en choisissant la couleur au travers des couleurs fondamentales en prenant en compte la luminosité présélectionnée. La couleur choisie sera la même sur tout l'afficheur.*
retroRouge() retroVert() retroBleu() retroOrange() retroViolet()	*Choix de couleurs prédéfinies pour le rétro-éclairage.*
allumer() eteindre()	*Active ou désactive le rétro-éclairage.*

Pour que ces méthodes puissent pleinement être exploitées avec l'objet correspondant, vous devez importer le module «touch.py » dans le répertoire où il se situe, c'est-à-dire « dothat » :

```
$ from dothat import touch
```

Voici ci-dessous le code source qui décrit l'ensembel de cette classe :

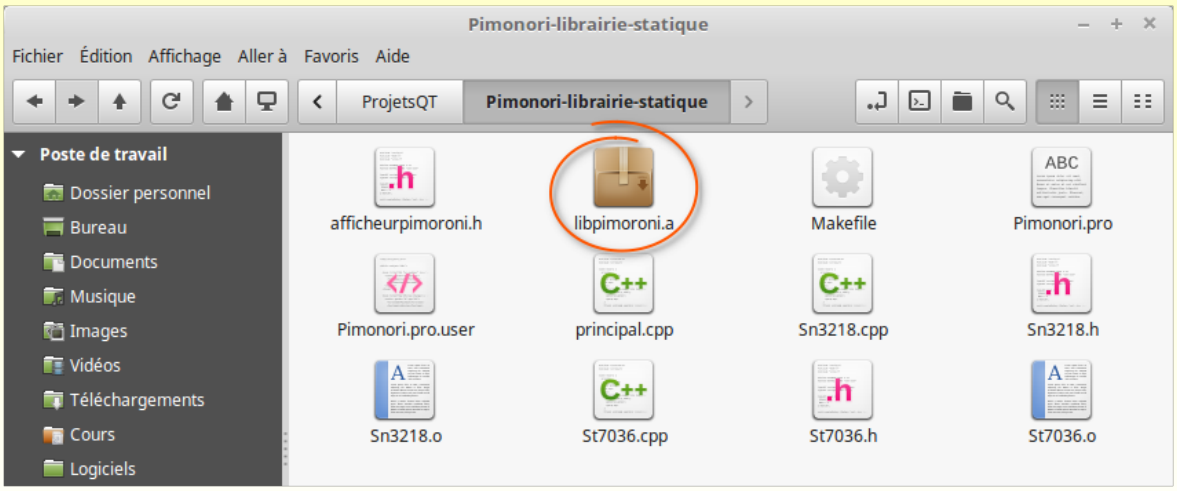
AfficheurPimoroni.h

```
#ifndef AFFICHEURPIMORONI_H
#define AFFICHEURPIMORONI_H

#include <wiringPi.h>
#include "St7036.h"
#include "Sn3218.h"
#include <string>
#include <map>
using namespace std;
```

```
class AfficheurPimoroni
{
  St7036 lcd;
  Sn3218 retro;
  int contraste = 50;
  map<char, char> accent = {{0xA9, 0x82}, {0xA2, 0x83}, {0xA0, 0x85},
                            {0xA7, 0x87}, {0xAA, 0x88}, {0xA8, 0x8A},
                            {0xAF, 0x8B}, {0xAE, 0x8C}, {0xB4, 0x93},
                            {0xBB, 0x96}, {0xB9, 0x97}};
public:
  int eclairage = 15;
private:
  string placerAccents(const string& texte)
  {
    string avecAccents;
    for (int i=0; texte[i]; i++)
      if (texte[i]<128) avecAccents += texte[i];
      else if (texte[i]==0xC3) { i++; avecAccents += accent[texte[i]]; }
    return avecAccents;
  }
public:
  AfficheurPimoroni()
  {
    wiringPiSetup();
    lcd.begin();
    lcd.home();
    lcd.setContrast(contraste);
    retro.begin();
    retro.enableLeds(Sn3218::CH_ALL);
    retroEclairage();
  }

  void afficher(const char* message, int ligne=0)
  {
    lcd.setCursorPosition(0, ligne);
    lcd.writeString(placerAccents(message).c_str());
  }

  void effacerEcran() { lcd.clear(); }

  void retroEclairage()
  {
    retro.enable();
    for (int i=0; i<18; i++) retro.set(i, eclairage);
    retro.update();
  }

  void retroEclairage(double rouge, double vert, double bleu)
  {
    double eclairage = this->eclairage*3 / (rouge+vert+bleu);
    retro.enable();
    for(int i = 0; i < 6; i++) {
      retro.set(i*3+0, bleu*eclairage);
      retro.set(i*3+1, vert*eclairage);
      retro.set(i*3+2, rouge*eclairage);
    }
    retro.update();
  }

  void retroRouge()   { retroEclairage(1, 0.05, 0.05); }
  void retroVert()    { retroEclairage(0.05, 1, 0.05); }
  void retroBleu()    { retroEclairage(0.05, 0.05, 1); }
  void retroOrange()  { retroEclairage(1, 0.5, 0); }
  void retroViolet()  { retroEclairage(1, 0.05, 1); }

  void changeContraste(int contraste)
  {
    this->contraste = contraste;
    lcd.setContrast(contraste);
  }

  void allumer()       { retro.enable(); }
  void eteindre()      { retro.disable(); }

  ~AfficheurPimoroni() { eteindre(); }
};
#endif // AFFICHEURPIMORONI_H
```

CRÉATION D'UNE LIBRAIRIE STATIQUE

Lorsque nous devons réaliser des projets qui prennent en compte cet afficheur, vous devez systématiquement intégrer d'une part la librairie « **wiringPi** » mais aussi tous les fichiers sources que nous venons de décrire.



*Afin d'éviter de les placer systématiquement dans chacun des projets que vous aurez à déployer, je vous invite à placer, dans votre ordinateur de développement, tous les fichiers en-tête dans le répertoire « **/usr/include** » et construite une librairie statique, avec les sources des deux classes de gestion de l'afficheur que vous placerez ensuite dans le répertoire « **/usr/lib** ».*

La compilation doit impérativement être réaliser pour une Raspberry, avec un processeur de type ARM-HF

pimoroni.pro

```
TEMPLATE = lib
CONFIG += c++11 staticlib
LIBS += -lwiringPi

SOURCES += Sn3218.cpp St7036.cpp
HEADERS += Sn3218.h St7036.h afficheurpimoroni.h

TARGET = pimoroni
```

EXEMPLE D'APPLICATION QUI EXPLOITE L'AFFICHEUR

Pour conclure sur ce sujet, je vous propose de réaliser une petite application déployée sur la Raspberry, qui prend en compte la librairie que nous venons de construire et le fichier en-tête relative à notre classe personnalisée **AfficheurPimoroni**.

pimoroni.pro

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle qt
LIBS += -lwiringPi -lpimoroni
SOURCES += principal.cpp
TARGET = pimoroni
target.path = /home/pi/c++
INSTALLS += target
```

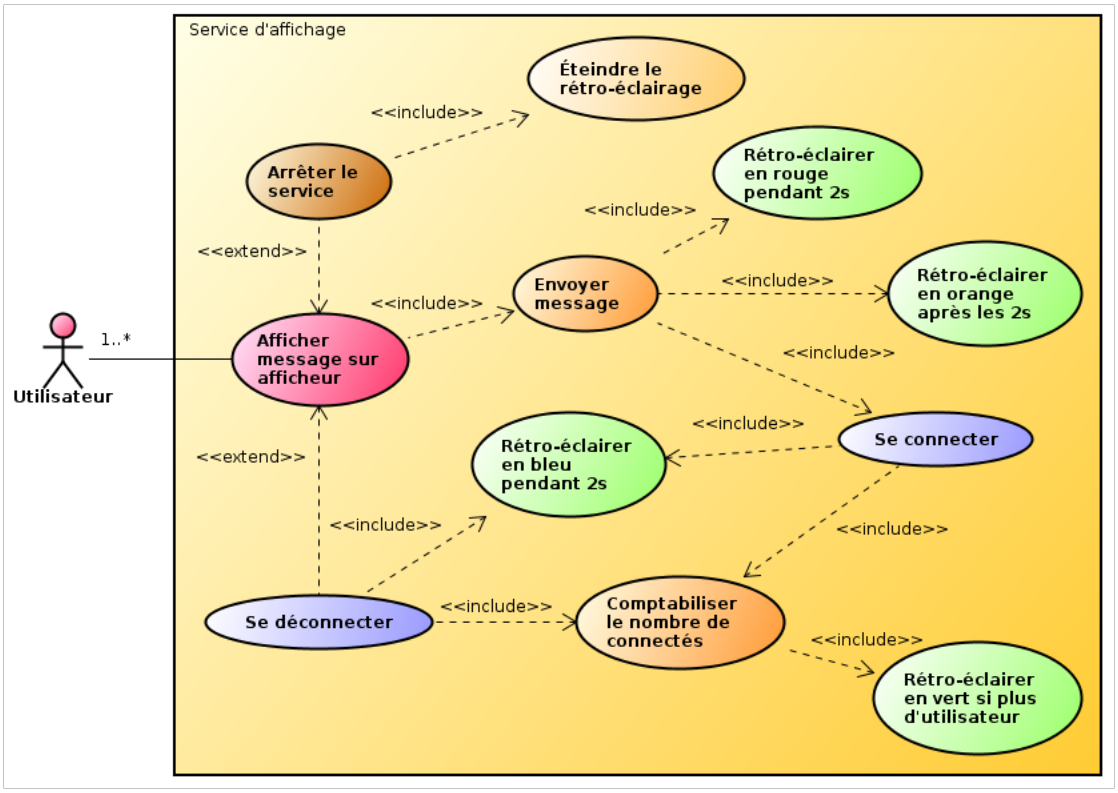
principal.cpp

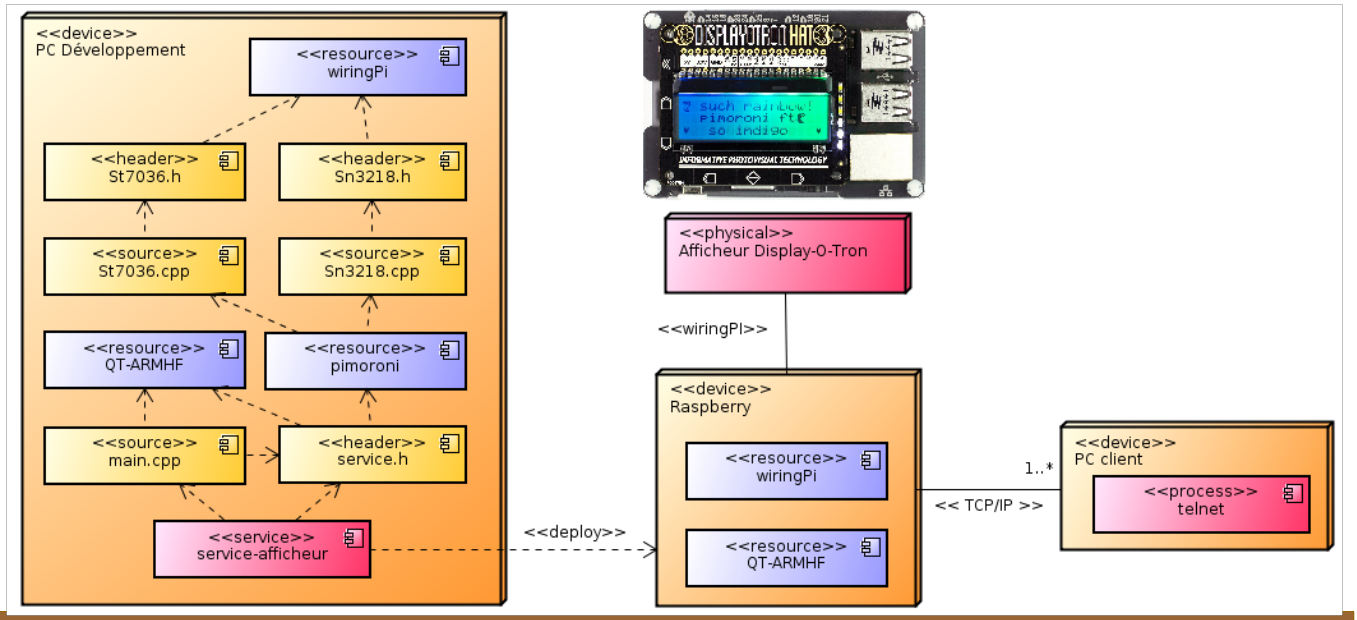
```
#include <afficheurpimoroni.h>

int main()
{
  AfficheurPimoroni ecran;
  ecran.afficher("Cet été c'est déjà le mois d'août");
  delay(3000);
  ecran.retroRouge();
  delay(3000);
  ecran.retroVert();
  ecran.effacerEcran();
  ecran.afficher("C'est la fête de rève");
  delay(3000);
  ecran.retroBleu();
  delay(3000);
  ecran.retroOrange();
  delay(3000);
  ecran.retroViolet();
  delay(3000);
  ecran.eteindre();
  delay(3000);
  ecran.allumer();
  delay(3000);
  return 0;
}
```

La fonction delay() est intégrée dans la librairie wiringPi.

SERVICE D'AFFICHAGE – COMMUNICATION DISTANTE

Pour conclure cette étude, nous allons mettre en œuvre un service sur la **Raspberry** qui nous permettra de décider du message à afficher depuis n'importe quel PC sur le réseau local, avec un simple client « **telnet** ». Nous profitons de l'occasion pour gérer le rétro-éclairage suivant les différentes situations tel que cela vous est spécifié dans le diagramme de cas d'utilisation :





pimoroni-reseau.pro

```
QT += core network
QT -= gui

TARGET = service-afficheur
CONFIG += console c++11
CONFIG -= app_bundle
LIBS += -lwiringPi -lpimoroni
TEMPLATE = app

SOURCES += main.cpp service.cpp
HEADERS += service.h

target.path = /home/pi/c++
INSTALLS += target
```

main.cpp

```
#include <QCoreApplication>
#include "service.h"

int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);
  Service afficheur;
  afficheur.connect(&afficheur, SIGNAL(quitter()), &a, SLOT(quit()));
  return a.exec();
}
```

service.h

```
#ifndef SERVICE_H
#define SERVICE_H

#include <QObject>
#include <QTcpServer>
#include <afficheurpimoroni.h>

class Service : public QObject
{
  Q_OBJECT
public:
  explicit Service(QObject *parent = 0);
signals:
  void quitter();
private slots:
  void nouvelleConnexion();
  void receptionMessage();
private:
  void deconnexion();
  void afficherNombre();
protected:
  void timerEvent(QTimerEvent *evt);
private:
  QTcpServer service;
  AfficheurPimoroni afficheur;
  int connectes=0;
};

#endif // SERVICE_H
```

service.cpp

```
#include "service.h"

#include <QTcpSocket>
#include <QTimerEvent>
#include <string>

using namespace std;

Service::Service(QObject *parent) : QObject(parent)
{
  connect(&service, SIGNAL(newConnection()), this, SLOT(nouvelleConnexion()));
  service.listen(QHostAddress::Any, 7070);
  afficheur.afficher("Service actif...");
}

void Service::nouvelleConnexion()
{
  QTcpSocket* client = service.nextPendingConnection();
  connect(client, SIGNAL(readyRead()), this, SLOT(receptionMessage()));
  QTextStream soumettre(client);
  soumettre << "Tapez votre message pour le visualiser sur l'afficheur" << endl;
  connectes++;
  afficheur.retroBleu();
  afficherNombre();
  startTimer(2000);
}

void Service::receptionMessage()
{
  QTcpSocket* client = (QTcpSocket*) sender();
  QString message = client->readAll();
  message = message.simplified();
  if (message=="quitter") {
    afficheur.eteindre();
    quitter();
  }
```

```
  else if (message=="stop")    {
    disconnect(client, SIGNAL(readyRead()), this, SLOT(receptionMessage()));
    afficheur.effacerEcran();
    client->close();
    deconnexion();
  }
  else {
    afficheur.effacerEcran();
    afficheur.afficher(message.toStdString().c_str());
    afficherNombre();
    afficheur.retroRouge();
    startTimer(2000);
  }
}

void Service::deconnexion()
{
  connectes--;
  afficheur.retroBleu();
  afficherNombre();
  if (!connectes) afficheur.retroVert();
  else startTimer(2000);
}

void Service::afficherNombre()
{
  string nombre = "Clients : ";
  nombre += to_string(connectes);
  afficheur.afficher(nombre.c_str(), 2);
}

void Service::timerEvent(QTimerEvent *evt)
{
  killTimer(evt->timerId());
  afficheur.retroOrange();
}
```