Projet en ISN

IUT GEII MARSEILLE
Patrick GUMUCHIAN

Lycée Alphonse Benoit – L'Isle sur la Sorgue
Marc SILANUS

2012 - 2013

Rev 2

SOMMAIRE

1 - Objectifs

1.1 - Position du problème

- Réaliser une application (logicielle et / ou matérielle) concrète à partir d'un cahier des charges avec un maximum d'autonomie.
- Des documents ressources seront mis à la disposition des élèves. Ces documents doivent fournir une aide pour trouver la solution au problème.

1.2 - Mise en situation de l'élève

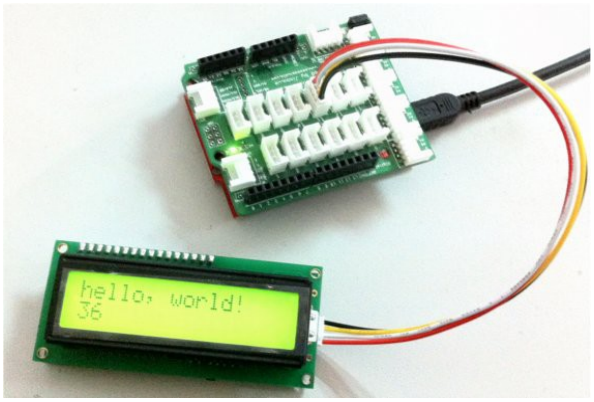
- Faire une démarche de réflexion pour **analyser** le problème :
  - Quelles connaissances théoriques manquent ils ?
- Faire une démarche de réflexion pour **trouver** une ou plusieurs solutions au problème :
  - Quels matériels, applications et logiciels peut on utiliser ?
- Faire une démarche de réflexion pour **choisir** une solution au problème :
  - Evaluer la difficulté pour réaliser la solution et prendre en compte aussi le critère économique.

2 - Exemples de Projets : Contrôle/commande

2.1 - Gestion d'un afficheur

2.1.1 - Cahier des charges

Proposer une application permettant de gérer un afficheur 16x2 via une liaison série.



Matériel :

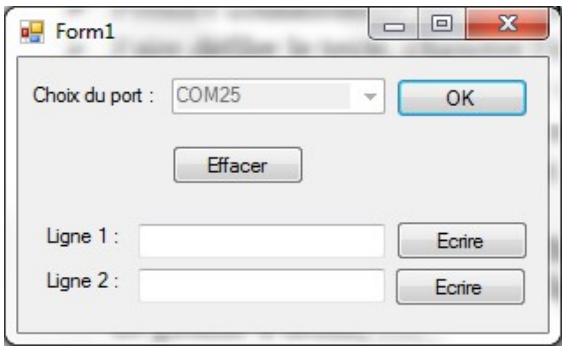
- Carte Arduino 1
- Carte Grove-Base Shield (interconnexions)
- Afficheur LCD 16x2

Logiciels :

- EDI Arduino
- Librairie SerialLCD
- Suite de développement PC

2.1.2 - Démarche à suivre

- Installer l'EDI Arduino 1.x (http://arduino.cc/en/main/software).
- Installer la librairie SerialLCD. (http://www.seeedstudio.com/wiki/File:SerialLCD_Library.zip).
- Ouvrir le programme d'exemple « SerialLCD/HelloWord » et l'analyser.
- Compiler et exécuter.
- Modifier le programme pour afficher « Bonjour le Monde ! ».
- Compiler et exécuter.
- Prendre connaissance des fonctions disponibles dans la libraire SerialLCD.
- Faire défiler le texte, clignoter l'écran, …
- Ouvrir le programme d'exemple « Communication/ASCIITable » et l'analyser.
- Compiler et exécuter. Ouvrir le moniteur série (dans le menu outils) et observer.
- Ecrire un programme permettant d'envoyer à l'afficheur le texte à écrire via une liaison série.
- Convenir d'un protocole pour effacer l'afficheur, pour écrire au début de la première ligne ou au début de la seconde ligne, pour faire défiler le texte de droite à gauche ou de gauche à droite, ….
- Ecrire un programme qui interprète ce protocole et gère l'afficheur en conséquence.
- Ecrire une application PC (interface Homme-Machine) qui gère l'afficheur connecté sur le port série choisi par l'utilisateur.



2.2 - Contrôle d'un projecteur DMX

2.2.1 - Cahier des charges

Proposer une application permettant de gérer un projecteur DMX 4 canaux à LED.

Matériel :

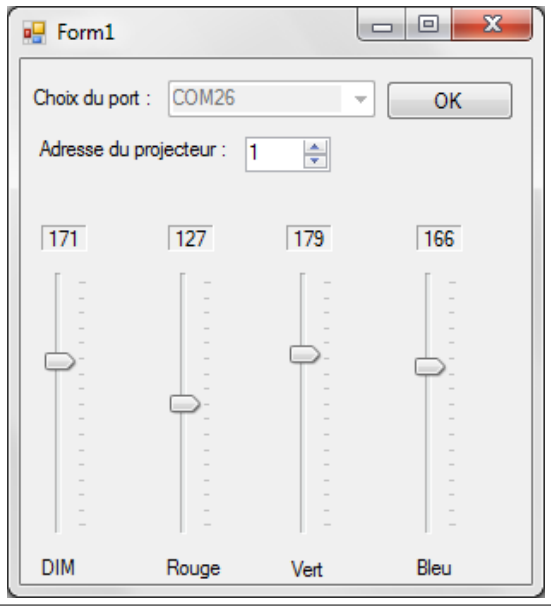
- Carte Arduino 1
- Shield DMX
- Projecteur DMX à LED 4 canaux
  - 1 : Dimmer
  - 2 : Rouge
  - 3 : Vert
  - 4 : Bleu

Logiciels :

- EDI Arduino
- Librairie DmxSimple
- Suite de développement PC

2.2.2 - Démarche à suivre

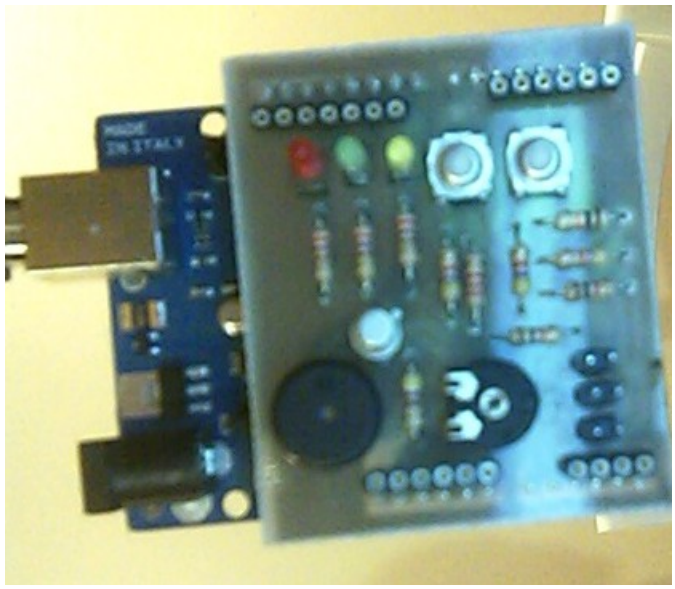
- Installer l'EDI Arduino 1.x (http://arduino.cc/en/main/software).
- Installer la librairie DmxSimple. (http://code.google.com/p/tinkerit/wiki/DmxSimple).
- Prendre connaissance des fonctions disponibles dans la libraire DmxSerial.
- La carte DMX utilise la broche 11 transmettre les données. Ouvrir le programme BasicDMX permettant d'envoyer au projecteur une succession de tableau Rouge/Vert/Bleu/Blanc.
- La carte DMX dispose d'un bouton poussoir sur la broche 7. Modifier le programme pour utilisé le bouton de la carte DMX afin de passer d'un tableau à l'autre.
- Ouvrir le programme SerialToDmx. Identifier les éléments de protocole. Excuter le programme et tester le protocole dans le moniteur série (menu outils).
- Ecrire une application PC (interface Homme-Machine) qui gère le projecteur connecté sur le port série choisi par l'utilisateur.



2.3 - Contrôle d'une carte de test Arduino

2.3.1 - Cahier des charges

Proposer une application permettant de contrôler une carte de test Arduino.



Matériel :

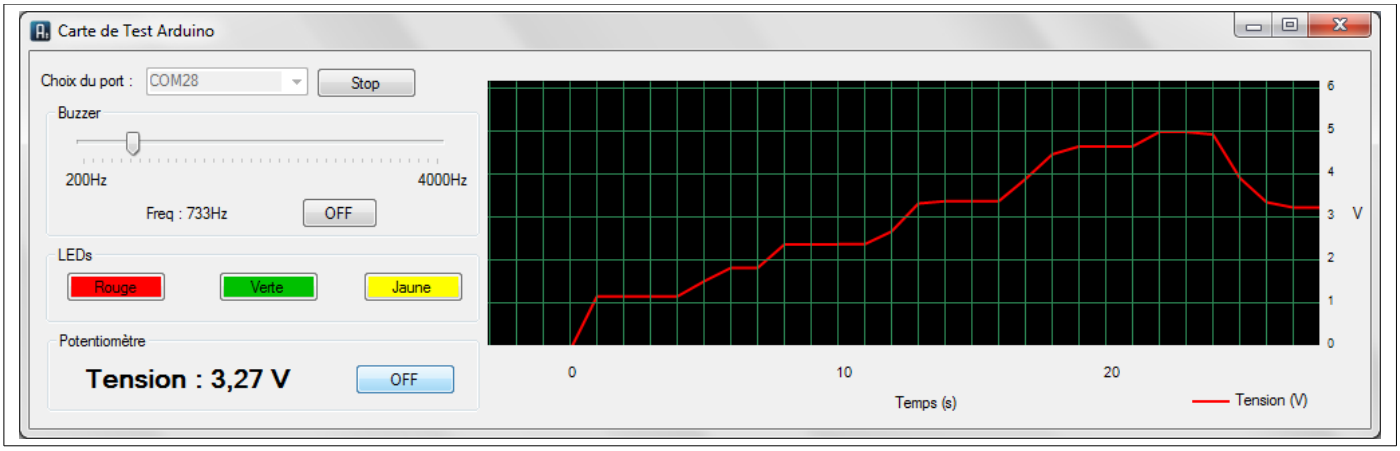
- Carte Arduino 1
- Shield Test comprenant :
  - 2 boutons poussoir
  - 1 poterntiomètre
  - 3 LEDs
  - 1 buzzer
  - 1 diviseur de tension (8 valeurs)

Logiciels :

- EDI Arduino
- Librairie DmxSimple
- Suite de développement PC

2.3.2 - Démarche à suivre

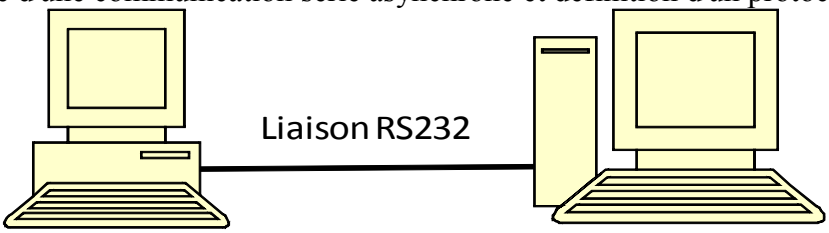
- Installer l'EDI Arduino 1.x (http://arduino.cc/en/main/software).
- Ouvrer le fichier CarteTest.
- Prendre connaissance des fonctions disponibles.
- Modifier le programme pour utiliser le bouton 1 afin d'allumer ou d'éteindre la led rouge.
- Modifier le programme pour utiliser le bouton 1 afin d'allumer successivement les trois leds, puis les éteindre toutes ensemble.
- Modifier le programme pour faire sonner le buzzer lorsqu'on appui sur le bouton 1.
- Ouvrir le programme carte_test_serie. Identifier les éléments de protocole. Excuter le programme et tester le protocole dans le moniteur série (menu outils).
- Ecrire une application PC (interface Homme-Machine) qui gère la carte connectée sur le port série choisi par l'utilisateur.



3 - Exemples de Projet : Communication sérielle

3.1 - Cahier des charges

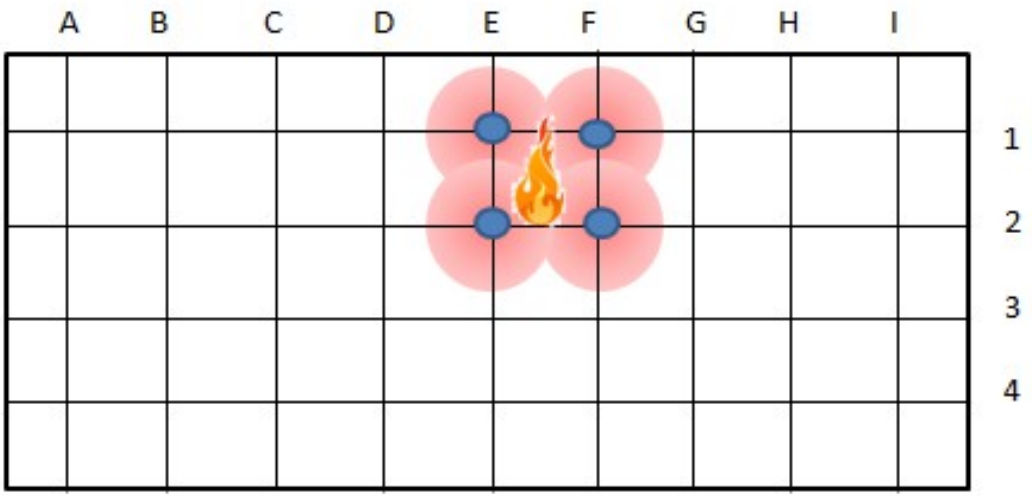
- Fournir une application de supervision permettant la localisation d'un incendie.
- Mise en œuvre d'une communication série asynchrone et définition d'un protocole.



PC / terminal
Simulation réseau de capteurs

PC / Superviseur
Localisation du foyer sur une carte
Communication avec le robot
Alerte / Appel d'urgence

Zone à surveiller



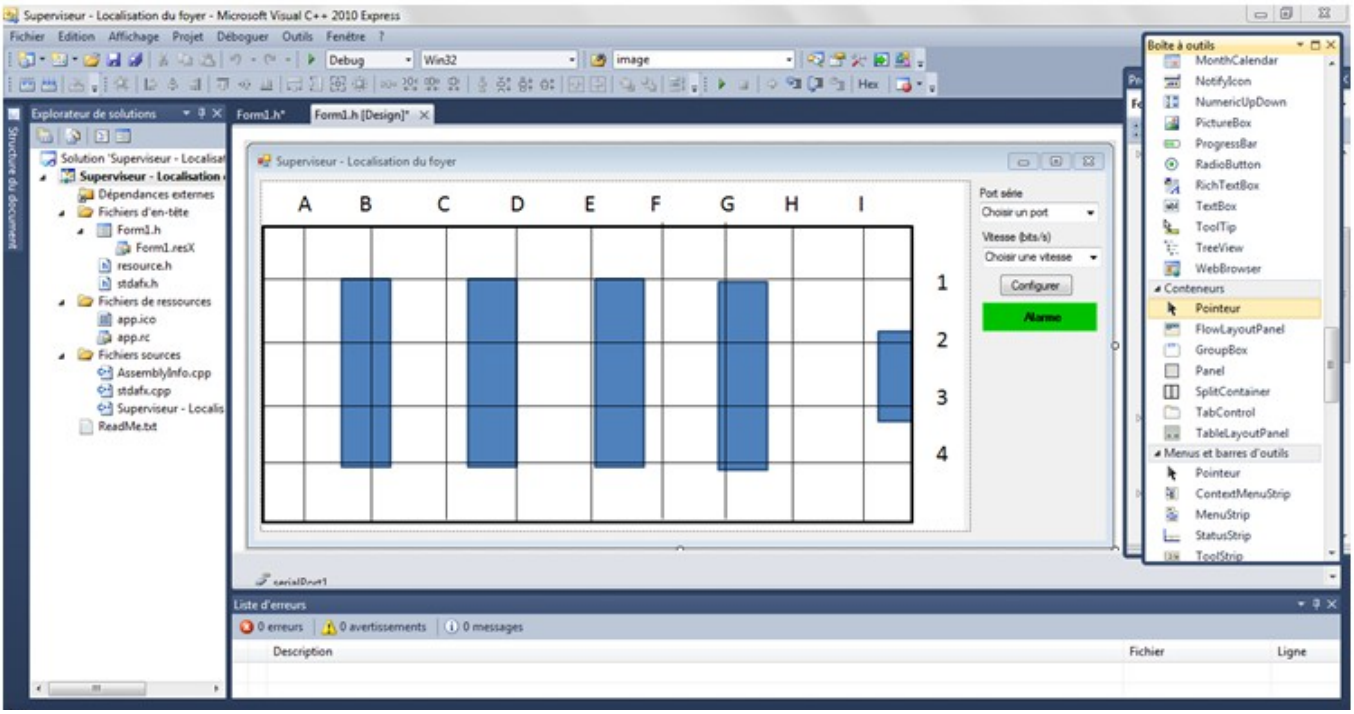
- Transmission des coordonnées des capteurs qui ont détecté un départ de feu : E1;F1;E2;F2;
- Protocole : transmission de 3 caractères **`Colonne Ligne ;`**

3.2 - Démarche à suivre

- Les connaissances théoriques :
  - Programmation Basic, C++, C#, Python, Java...
  - Protocole de communication RS232.
- Les outils et le matériel nécessaire :
  - Un logiciel de développement rapide (IDE : Integreted Developpement Environement) : Delphi, BuilderC++, Visual Studio, Qt Creator, ...
  - 2 PC avec un logiciel de terminal : Hyperterminal, Putty, Minicom, terminal, ...

4 - Travail à réaliser

- Relier 2 PC entre-eux en utilisant un câble série. Si les PC ne disposent pas de ports séries, utiliser un adaptateur USB/Série.
- Ouvrir un terminal sur les PC, configurer la connexion selon votre choix et tester le transfert de données.
- Choisir un outil de développement et remplacer un terminal par votre application de supervision.



- Appeler un formateur si vous souhaitez visualiser le signal de la liaison série RS232.

Remarque : Ce projet peut être adapter pour réaliser une application de jeu comme le morpion ou la bataille navale. On devra alors réaliser une application permettant de gérer l'émission et la réception de données.

5 - Exemples de Projet : Traitement d'image

5.1 - Cahier des charges

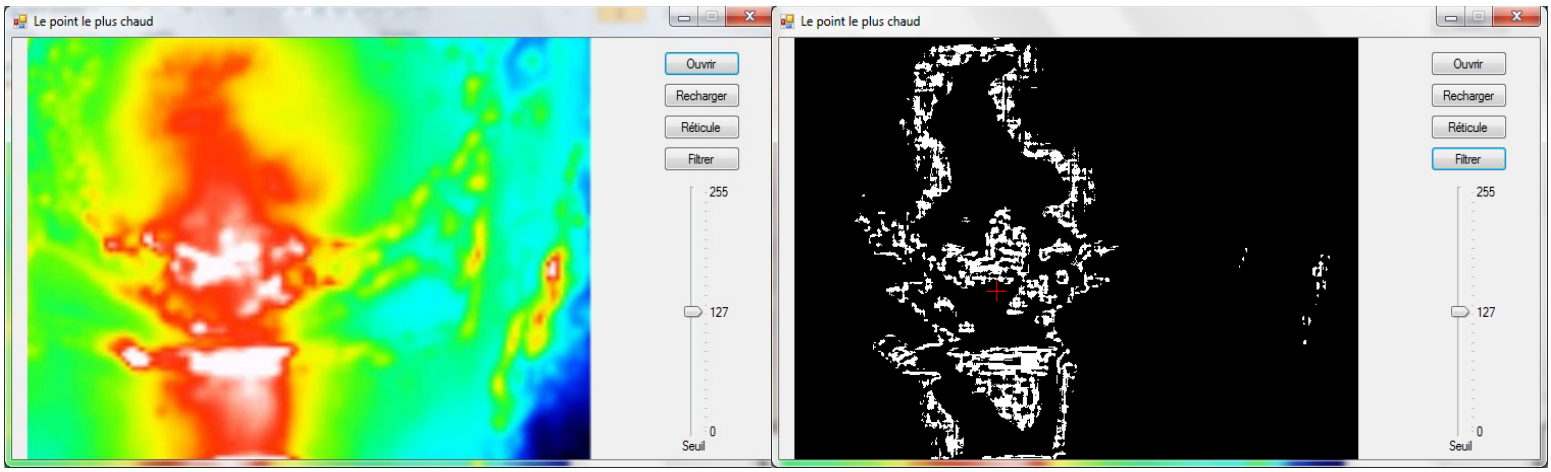
- Fournir une application de traitement d'image permettant d'identifier le centre du foyer d'un incendie.
- L'application doit permettre de :
  - Charger une image issue d'une caméra thermique.
  - Régler le seuil de sensibilité de température.
  - Matérialiser le centre du foyer (calcul du barycentre).

5.2 - Démarche à suivre

- Les connaissances théoriques :
  - Constitution d'une image BMP.
  - Algorithme de calcul du barycentre.
- Les outils et le matériel nécessaire :
  - Un logiciel de développement rapide (IDE : Integreted Developpement Environement) : Delphi, BuilderC++, Visual Studio, Qt Creator, ...
  - Une caméra thermique ou à défaut, plusieurs images thermiques stockées sur le PC.

6 - Travail à réaliser

- Ouvrir une image thermique dans Paint et identifier les valeurs des couleurs représentatives d'un point chaud.
- Concevoir l'application permettant de charger une image à l'écran.
- Ajouter un bouton pour filtrer les couleurs Rouge, Vert et Bleu selon l'algorithme suivant :
  - Si ComposanteRouge = 255 et si ComposanteVerte > 127 alors CouleurPixel = Blanc
  - Sinon CouleurPixel = Noir.
- Ajouter un curseur pour régler la valeur du seuil de sensibilité de la température (proportion de vert dans l'image) et modifier l'algorithme précédant pour tenir compte de se réglage.
- Calculer et afficher une petite croix au centre du foyer.



7 - Annexes Arduino

Références complètes du langage : **http://arduino.cc/en/Reference/HomePage**

7.1 - Structure d'un programme

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

// Initialization
void setup()
{
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

// maim program, loop without end
void loop()
{
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH)
  {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

7.2 - Principales fonctions

`pinMode(pinNumber, Direction)` : Définit la direction de la broche **`pinNumber`**. Le paramètre **`Direction`** peut prendre les valeurs **`INPUT`**, **`OUTPUT`** ou **`INPUT_PULLUP`** (résistance de tirage intégrée).

`digitalWrite(pinNumber, logicState)` : Ecrit sur la broche **`pinNumber`** l'état logique **`logicState`** qui peut prendre les valeurs **`HIGH`** ou **`LOW`**.

`digitalRead(pinNumber)` : Lit sur la broche **`pinNumber`** l'état logique présent. Retourne un octet (**`byte`**) à **`HIGH`** ou **`LOW`** (1 ou 0).

`analogRead(pinNumber)` : Lit sur la broche **`pinNumber`** la valeur numérique correspondante à la conversion analogique/numérique de la tension présente. Retourne un entier (**`int`**) compris entre 0 et 1023.

7.3 - Ajouter une librairie

L'ajout de libraire se fait au moyen de la directive de compilation **`#include`** . Les librairies sont installées dans le dossier **`arduino-X.XX\libraries`**

```
#include <DmxSimple.h>
```

7.4 - La librairie Serial

La librairie Serial est utilisée pour les communications par le port série entre la carte Arduino et un ordinateur ou d'autres composants. Toutes les cartes Arduino ont au moins un port Série (également désigné sous le nom de UART ou USART) : Serial. Ce port série communique sur les broches 0 (**RX**) et 1 (**TX**) avec l'ordinateur via le port USB.

7.4.1 - Les fonctions

begin() : Fixe le débit de communication en bits par secondes (l'unité est le baud) pour la communication série.

Exemple :

```
Serial.begin(9600) ; //Définit le débit de la liaison à 9600 bits/sec (bauds)
```

available() : Donne le nombre d'octets (caractères) disponible pour lecture dans la file d'attente (buffer) du port série.

read() : Retourne le premier octet disponible sur le port Série ou -1 si aucune donnée.

print() : Ecrit les données sur le port série sous forme lisible pour les humains (texte ASCII). Cette instruction peut prendre plusieurs formes :

```
Serial.print(78); // affiche "78"
Serial.print(1.23456); // affiche "1.23"
Serial.print(byte(78)); // affiche "N" (valeur ASCII : 78)
Serial.print('N'); // affiche "N"
Serial.print("Hello world."); // affiche "Hello world."
Serial.print(78, BYTE); // affiche "N"
Serial.print(78, BIN) ; // affiche  "1001110"
Serial.print(78, OCT); // affiche "116"
Serial.print(78, DEC); // affiche "78"
Serial.print(78, HEX); // affiche "4E"
Serial.println(1.23456, 0); // affiche "1"
Serial.println(1.23456, 2); // affiche "1.23"
Serial.println(1.23456, 4); // affiche "1.2346"
```

println() : Identique à print et ajoute un caractère de "retour de chariot" (ASCII 13, or '**\r**') et un caractère de "nouvelle ligne" (ASCII 10, or '**\n**') à la fin des données à transmettre.

7.4.2 - Exemple

```
// envoie des données seulement quand vous recevez des données :
if (Serial.available() > 0) // si des données entrantes sont présentes
{
	// lit le 1er octet arrivé
	int incomingByte = Serial.read();

	// dit ce que vous obtenez
	Serial.print("J'ai recu : ");
	Serial.println(incomingByte, DEC);
}
```

8 - Annexes Python

8.1 - Le module pyserial

Site web officiel : **http://pyserial.sourceforge.net/**

8.1.1 - Importer le module

```
>>> import serial
```

8.1.2 - Construction d'un objet serial

Configuration par défaut :

- 9600 bauds,
- 8 bits de données,
- 1 bit de stop,
- pas de parité,
- pas de time out,
- pas de contrôle de flux.

```
>>> ser = serial.Serial(0)  # open first serial port

>>> ser = serial.Serial('/dev/ttyS1') # Linux

>>> ser = serial.Serial('COM1') # Windows

>>> ser = serial.Serial(0, 38400, timeout=0, parity=serial.PARITY_EVEN,
    rtscts=1)
```

8.1.3 - Ouverture du port

```
>>> ser.open()
```

8.1.4 - Fermeture du port

```
>>> ser.close()
```

8.1.5 - Lire des données sur le port série

```
>>> inByte ser.read()
>>> line = ser.readline() # chaîne terminée par \n
```

8.1.6 - Ecrire des données sur le port série

```
>>> ser.write("hello") # Ecrire une chaîne
>>> ser.write('\x01')  # Ecrire une valeur en hexadécimal
```

8.2 - Exemple

```
import serial
import time
ser = serial.Serial("COM2")   # open serial port
time.sleep(2)                 # wait for opening port
ser.write("hello")            # write a string
ser.close()                   # close port
```

9 - Annexes Visual C#

Les composants et classes suivants sont issues de Visual Studio Express. Les exemple de code sont en C# mais peuvent facilement être traduit dans d'autres langages.

9.1 - Le composant SerialPort

Représente une ressource de port série.

9.1.1 - Principales propriétés

`Name` : Nom de l'objet instancié.

`BaudRate` : Obtient ou définit la vitesse en bauds série.

`DataBits` : Obtient ou définit la longueur standard des bits de données par octet.

`Parity` : Obtient ou définit le protocole de contrôle de parité.

`StopBits` : Obtient ou définit le nombre standard de bits d'arrêt par octet.

`Handshake` : Obtient ou définit le type de contrôle de flux pour la transmission de données par le port série.

`PortName` : Obtient ou définit le port pour les communications, y compris, de manière non limitative, tous les ports COM disponibles.

9.1.2 - Principales méthodes

`Open()` : Ouvre une nouvelle connexion au port série.

`Close()` : Ferme la connexion au port série.

`ReadTo(String value)` : Lit une chaîne jusqu'au value spécifié. Retourne une chaine de caractères.

`ReadExisting()` : Lit tous les octets immédiatement disponibles, en fonction de l'encodage, dans le flux et la mémoire tampon d'entrée de l'objet SerialPort. Retourne une chaine de caractères.

`Write(String text)` : Écrit la chaîne spécifiée au port série.

9.1.3 - Principal événement

`DataReceived` : Représente la méthode qui gérera l'événement reçu avec les données d'un objet SerialPort.

ex :

```
private: Void sp_DataReceived(Object  sender,SerialDataReceivedEventArgs  e)
{
        String indata = sp.ReadExisting();
        Label.Text = "Données reçues :" + indata;
}
```

9.2 - Le composant OpenFileDialog

Permet d'afficher une boite de dialogue qui invite l'utilisateur à ouvrir un fichier.

9.2.1 - Principales propriétés

`Name` : Nom de l'objet instancié.

`Filter` : Définit la chaîne de filtrage des noms de fichiers en cours
ex: **`Text Files (*.txt)|*.txt`**

`DefaultExt` : Définit l'extension de nom de fichier par défaut.
ex: **`txt`**

`FileName` : Obtient ou définit une chaîne (de type **`String`**) comportant le nom de fichier sélectionné dans la boîte de dialogue d'ouverture de fichier.

9.2.2 - Principale méthode

`ShowDialog()` : Exécute une boîte de dialogue standard d'ouverture de fichier.
Retourne le choix de l'utilisateur (OK ou Annuler). La valeur de retour est de type **`System.Windows.Forms::DialogResult`** et les valeurs possibles sont :

- **`Cancel`**
- **`OK`**

ex :

```
if(openFileDialog.ShowDialog() == DialogResult.OK)
{
	//Do something
}
```

9.3 - La classe Graphics

Encapsule une surface de dessin GDI+ (Graphic Device Interface). Nécessaire pour afficher une image dans la fenêtre de l'application.

9.3.1 - Espace de nom

```
Using namespace System.Drawing;
```

9.3.2 - Définition

```
private Graphics graph; //Définition de l'objet graph
```

9.3.3 - Principales méthodes

`CreateGraphics()`: Création de l'objet Graphics. A exécuter en général au chargement de la fenêtre.
ex : **`graph = CreateGraphics;`**

`DrawImage(Image, Int32, Int32)` : Dessine l'image spécifiée, en utilisant sa taille physique d'origine, à l'emplacement indiqué par une paire de coordonnées.

ex :

```
// Create image.
Image newImage = Image.FromFile( "SampImag.jpg" );

// Create coordinates for upper-left corner of image.
int x = 100;
int y = 100;

// Draw image to screen.
graph.DrawImage( newImage, x, y );
```

`DrawArc(Pen pen, int x, int y, int width, int height, int startAngle, int sweepAngle)` :

Dessine un arc représentant une partie d'une ellipse spécifiée par une paire de coordonnées, une largeur et une hauteur.

ex :

```
// Create pen.
Pen blackPen = new Pen( Color.Black,3.0f );
// Create coordinates of rectangle to bound ellipse.
int x = 40;
int y = 40;
int width = 100;
int height = 200;
// Create start and sweep angles on ellipse.
int startAngle = 45;
int sweepAngle = 270;

// Draw arc to screen.
Graph.DrawArc( blackPen, x, y, width, height, startAngle, sweepAngle );
```

`Clear(Color color)` : Efface l'intégralité de la surface de dessin et la remplit avec la couleur d'arrière-plan spécifiée.
Color est une structure qui représente la couleur d'arrière-plan de la surface de dessin. Une couleur est représentée par son code ARVB (alpha, rouge, vert, bleu).

ex :

```
//Clear screen with teal background.
Graph.Clear(Color.Teal)

//Clear screen with default form background.
Graph.Clear(this.BackColor);
```

9.4 - La classe Bitmap

Encapsule une bitmap GDI+, composée des données de pixels d'une image graphique et de ses attributs.

9.4.1 - Espace de nom

```
Using namespace System.Drawing;
```

9.4.2 - Définition

```
private Bitmap   bmp; //Définition de l'objet bmp
```

9.4.3 - Constructeur

`Bitmap(String filename)` : Initialise une nouvelle instance de la classe Bitmap à partir du fichier spécifié.

ex :

```
String strPicture = openFileDialog.FileName;
bmp = new Bitmap(strPicture);
```

9.4.4 - Principales propriétés

`Height` : Obtient la hauteur du Bitmap, en pixels.

`Width` : Obtient la hauteur du Bitmap, en pixels.

9.4.5 - Principales méthodes

`GetPixel(int x, int y)` : Obtient la couleur du pixel spécifié dans le Bitmap. Retourne une structure Color représentant la couleur du pixel spécifié. Une couleur est représentée par son code ARVB (alpha, rouge, vert, bleu).

ex :

```
for ( int col=0 ; col<bmp.Width ; col++)
{
       for ( int row=0 ; row<bmp.Height ; row++)
       {
            // Rechercher les pixel Rouges
            if( bmp.GetPixel(col,row).R==255 &&
                bmp.GetPixel(col,row).G==0   &&
                bmp.GetPixel(col,row).B==0)
            {
                  //Do something
            }
        }
}
```

`SetPixel(int x, int y)` : Définit la couleur du pixel spécifié dans le Bitmap. Une couleur est représentée par son code ARVB (alpha, rouge, vert, bleu).

ex :

```
for ( int col=0 ; col<bmp.Width ; col++)
{
       for ( int row=0 ; row<bmp.Height ; row++)
       {
            // Rechercher les pixel Rouges et les changer en Vert
            if( bmp.GetPixel(col,row).R>200 &&
                bmp.GetPixel(col,row).G==0   &&
                bmp.GetPixel(col,row).B==0)
            {
                  bmp.SetPixel(col, row, Color.Green);
            }
        }
}
```