

TABLE DES MATIERES

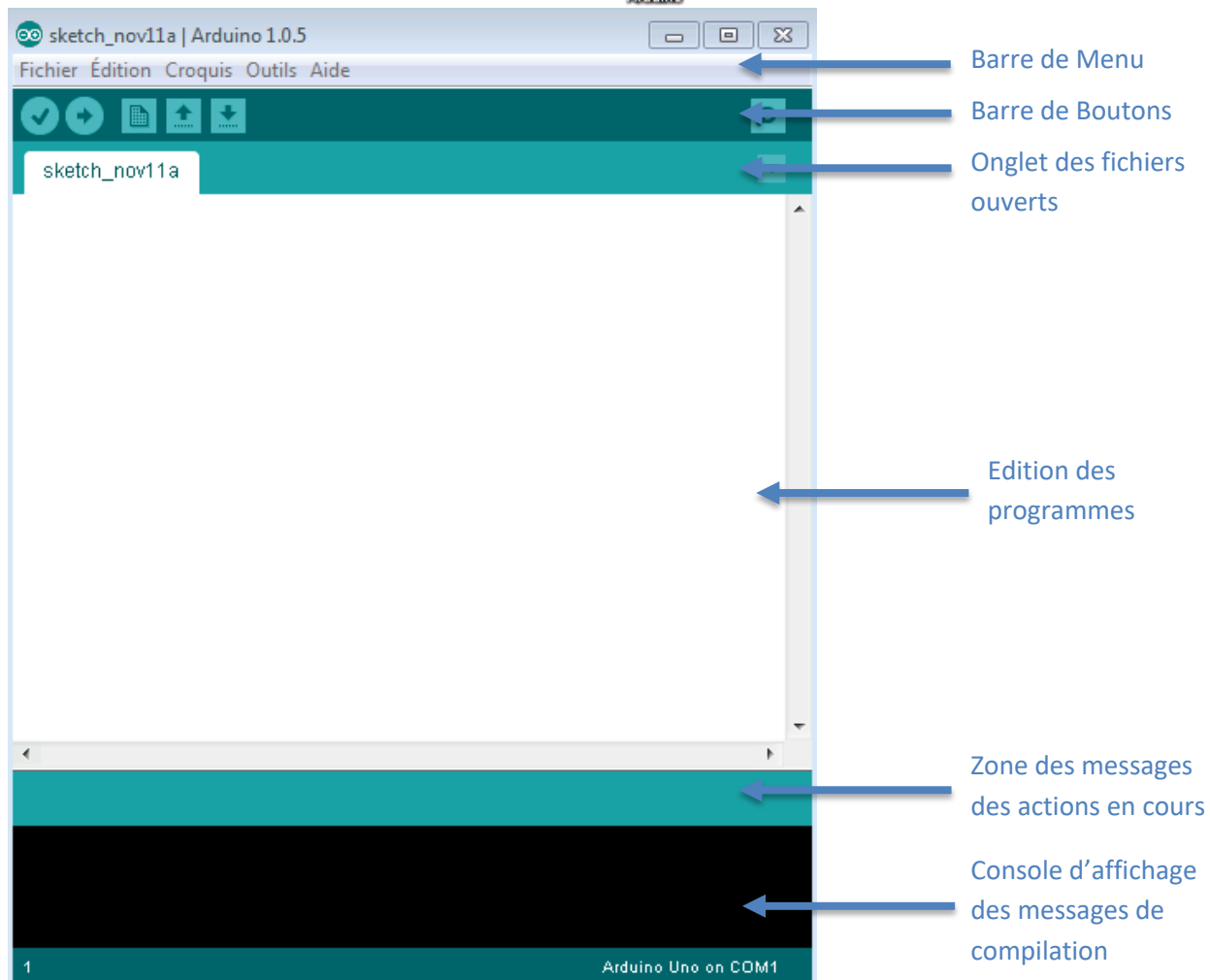
1	Interface de programmation	2
2	Structure d'un programme	3
3	Commentaires	4
4	Données, variables et constantes	4
5	Opérateurs	5
6	Fonctions	6
6.1	Fonctions arithmétiques	6
6.2	Fonctions de manipulation de bits	6
6.3	Fonctions de gestion du temps	7
6.4	Fonctions de gestion des E/S numériques	7
6.5	Fonctions de gestion des sorties numériques PWM	7
6.6	Fonctions de gestion des entrées analogiques	7
6.7	Fonctions particulières de gestion des E/S	8
6.8	Fonctions de gestion du port série asynchrone	8
7	Bibliothèques	10
8	Développement d'un projet	11

1 INTERFACE DE PROGRAMMATION

Le logiciel IDE Arduino a pour fonctions principales :

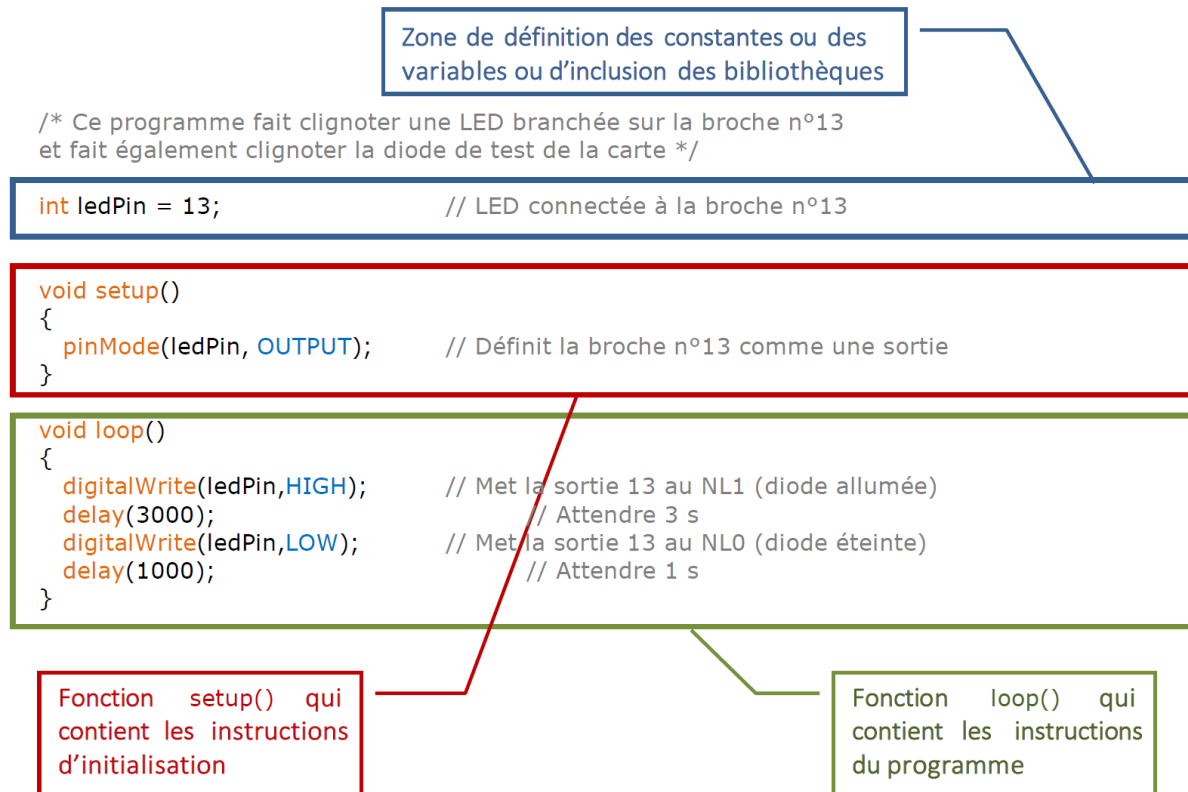
- De pouvoir écrire et compiler des programmes pour la carte Arduino
- De se connecter avec la carte Arduino pour y téléverser (transférer) les programmes
- De communiquer avec la carte Arduino

Pour ouvrir l'IDE d'Arduino, il faut utiliser l'icône :



2 STRUCTURE D'UN PROGRAMME

Un programme ou croquis (*sketch*) destiné à une carte Arduino est constitué de 3 parties.



La première partie permet de définir les constantes et les variables en déclarant leur type. Elle permet également l'inclusion des bibliothèques utilisées dans le programme au moyen de `#include`.

La fonction `setup()` contient les instructions d'initialisation ou de configuration des ressources de la carte comme par exemple, la configuration en entrée ou sorties des broches d'E/S numériques, la définition de la vitesse de communication de l'interface série, etc.. Cette fonction est exécutée qu'une seule fois juste après le lancement du programme.

La fonction `loop()` contient les instructions du programme à proprement parlé. Cette fonction sera répétée indéfiniment tant que la carte Arduino restera sous tension.

3 COMMENTAIRES

Pour placer des commentaires sur une ligne unique ou en fin de ligne, il faut utiliser la syntaxe suivante :

```
// Cette ligne est un commentaire sur UNE SEULE ligne
```

Pour placer des commentaires sur plusieurs lignes :

```
/* Commentaire, sur PLUSIEURS lignes qui sera ignoré par le programme,
mais pas par celui qui lit le code
*/
```

4 DONNEES, VARIABLES ET CONSTANTES

Les différents types utilisés avec la programmation Arduino sont :

Nom	Description	Exemples
boolean	Donnée logique (true ou false) sur 8 bits	boolean marche = true ;
byte	Octet	byte a = 145 ; byte a = B00010010 ;
int	Entier signé sur 16 bits	int b = -2896
unsigned int	Entier non signé sur 16 bits	unsigned int c = 45768
long	Entier signé sur 32 bits	long d = 12345678
unsigned long	Entier non signé sur 32 bits	unsigned long e = 418755889
float	Nombre à virgule flottant sur 32 bits	float f = 2.5891
char	Caractère sur 8 bits, caractères du code ASCII étendu (valeur -128 à 127)	char lettreA = 'A' ;
unsigned char	Caractère sur 8 bits, caractères du code ASCII étendu (valeur 0 à 255)	char lettreA = 65 ;

Une constante peut être définie au moyen de const ou de #define :

```
#define N 2 //Toute variable N rencontrée dans le programme sera
           // remplacée par la valeur 2
const byte N=2 // N est une constante de type byte et de valeur 2
```

Un certain nombre de noms de constantes sont prédéfinis :

Nom	Description
true	Donnée binaire égale à 1 ou donnée numérique différente de 0
false	Donnée binaire ou numérique différente de 1
HIGH	Génération d'un niveau de tension haut sur une des sorties numériques
LOW	Génération d'un niveau de tension bas sur une des sorties numériques
INPUT	Configuration d'une broche numérique en entrée
OUTPUT	Configuration d'une broche numérique en sortie

5 OPERATEURS

Opérateurs arithmétiques		
+	Addition	$c = a + b ;$
-	Soustraction	$c = a - b ;$
*	Multiplication	$c = a * b ;$
/	Division entière	$c = a / b ;$
%	Reste de la division entière	$c = a \% b ;$

Il existe une notation rapide si une variable est présente à droite et à gauche de l'opérateur :

$a = a + b ;$ peut être noté $a += b ;$

Décrémentations, incréments		
++	Décrémentations (retirer 1)	$a-- ;$
--	Incrémentations (ajouter 1)	$a++ ;$

Opérateurs binaires		
&	ET bit à bit	$c = a \& b$
	OU bit à bit	$c = a b$
^	OU exclusif	$c = a ^ b$
!	NON	$b = ! a$
>> <i>n</i>	Décalage à droite de <i>n</i> bits	$c = a >> 3 ;$
<< <i>n</i>	Décalage à gauche de <i>n</i> bits	$c = a << 3 ;$

Opérateurs binaires

&&	ET logique	if (a && b)
	OU Logique	if (a b)
==	Égal	if (a == b)
!=	Différente	if (a != b)
>	Supérieur	if (a > b)
>=	Supérieur ou égal	if (a >= b)
<	Inférieur	if (a < b)
<=	Inférieur ou égal	if (a <= b)

6 FONCTIONS

6.1 FONCTIONS ARITHMETIQUES

Fonctions arithmétiques prédéfinies

min(x, y)	Cette fonction retourne la valeur la plus petite entre les 2 variables
max(x, y)	Cette fonction retourne la valeur la plus grande entre les 2 variables
contrain(x, a, b)	Cette fonction impose une plage de variation à la variable x comprise entre les bornes a et b. Cette fonction peut être utilisée pour limiter la tension fournie par un capteur
abs(x)	Cette fonction retourne la valeur absolue de la variable
sort(x)	Cette fonction retourne la racine carrée de la variable
map(x, min, max, newmin, newmax)	Cette fonction permet de réaliser une translation de la plage de variation de la variable

6.2 FONCTIONS DE MANIPULATION DE BITS

Fonctions de manipulation de bits

lowByte(x)	Cette fonction permet d' extraire les 8 bits de poids faible de la variable La valeur retournée est de type byte alors que le paramètre peut être de n'importe quel type
highByte(x)	Cette fonction permet d' extraire les 8 bits de poids fort de la variable
bitRead(x, n)	Cette fonction permet de lire le bit n de la variable
bitWrite(x, n, Level)	Cette fonction permet de modifier le bit n de la variable en fonction du paramètre Level
bitSet(x, n)	Cette fonction permet de mettre à 1 le bit n de la variable

<code>bitClear(x, n)</code>	Cette fonction permet de mettre à 0 le bit n de la variable
-----------------------------	---

6.3 FONCTIONS DE GESTION DU TEMPS

Fonctions de gestion du temps	
<code>delay(n)</code>	Cette fonction réalise une pause de n ms. Le paramètre n est du type <code>unsigned long</code>
<code>delayMicroseconds(n)</code>	Cette fonction réalise une pause de n μ s. Le paramètre n est du type <code>unsigned int</code>

6.4 FONCTIONS DE GESTION DES E/S NUMERIQUES

Fonctions de gestion des E/S numériques	
<code>pinMode(num, sens)</code>	Cette fonction configure la broche numéro num dans la direction $sens$ (OUTPUT, INPUT)
<code>digitalWrite(num, Level)</code>	Cette fonction permet d' imposer un niveau logique haut (HIGH) ou bas (LOW) sur la sortie numérique sélectionnée
<code>digitalRead(num)</code>	Cette fonction permet de lire le niveau logique sur l'entrée numérique sélectionnée. Cette fonction ne retourne que deux valeurs HIGH ou LOW.

6.5 FONCTIONS DE GESTION DES SORTIES NUMERIQUES PWM

Les cartes Arduino sont capables de générer des signaux à **Modulation à Largeur d'Impulsions (MLI)** ou *Pulse Width Modulation (PWM)*. Il s'agit de signaux logiques rectangulaires de fréquence fixe égale à 490 Hz et à rapport cyclique programmable.

Fonctions de gestion des sorties analogiques	
<code>analogWrite(num, duty)</code>	<p>Cette fonction permet de générer un signal PWM sur la sortie sélectionnée avec le rapport cyclique désiré. La paramètre $duty$ doit être de type <code>byte</code>, c'est-à-dire compris entre 0 et 255 pour un rapport cyclique compris entre 0 et 100%.</p> <p>Le signal PWM est généré à partir de l'exécution de la fonction <code>analogWrite()</code> et jusqu'à ce qu'une nouvelle fonction sur la broche soit appelée : <code>analogWrite()</code>, <code>digitalRead()</code> ou <code>digitalWrite()</code></p>

6.6 FONCTIONS DE GESTION DES ENTREES ANALOGIQUES

Les entrées analogiques sont connectées à un **convertisseur analogique numérique (CAN) 10 bits**. La sortie du CAN génère une donnée égale à **0** lorsque la **tension d'entrée est nulle** et une donnée égale à **1023** lorsque la **tension d'entrée est égale à la tension de référence** du CAN.

Fonctions de gestion des entrées analogiques

<code>analogReference(<i>type</i>)</code>	<p>Cette fonction permet de définir la tension de référence du CAN. Le paramètre <i>type</i> peut prendre les valeurs suivantes :</p> <ul style="list-style-type: none"> ▪ DEFAULT : La tension de référence est égale à la tension d'alimentation de la carte Arduino (5V pour Arduino Uno) ▪ INTERNAL : La tension de référence est égale à 1,1V pour Arduino Uno ▪ EXTERNAL : La tension de référence est égale à la tension appliquée sur l'entrée externe AREF de la carte Arduino.
<code>analogRead(<i>num</i>)</code>	<p>Cette fonction permet de lire le résultat de la conversion analogique numérique de la tension présente sur l'entrée analogique sélectionnée</p> <p>La valeur retournée est du type <code>int</code> est comprise entre 0 et 1023</p>

6.7 FONCTIONS PARTICULIERES DE GESTION DES E/S

Fonctions particulières

<code>tone(<i>num</i>, <i>freq</i>, <i>duration</i>)</code>	<p>Cette fonction permet de générer un signal carré sur la sortie sélectionnée (rapport cyclique de 50%, fréquence programmable et durée programmable)</p> <p>Le paramètre <i>freq</i> est exprimé en Hz et le paramètre <i>duration</i> en ms. Si la durée n'est pas précisée, le signal est généré jusqu'à ce que la fonction <code>notone()</code> soit exécutée.</p>
<code>notone(<i>num</i>)</code>	<p>Cette fonction permet de stopper la génération du signal carré sur la sortie sélectionnée</p>
<code>pulseIn(<i>num</i>, <i>type</i>, <i>delaymax</i>)</code>	<p>Cette fonction permet de mesurer la durée d'une impulsion sur l'entrée sélectionnée. Le résultat retourné est du type <code>unsigned long</code></p> <p>Le paramètre <i>type</i> peut être HIGH dans le cas d'une impulsion au niveau 1 ou LOW d'une impulsion au niveau 0.</p> <p>Le paramètre <i>delaymax</i> permet de définir le délai d'attente de l'impulsion. Lorsque ce délai est dépassé et qu'aucune impulsion ne s'est produite, la fonction retourne une valeur nulle. Ce paramètre est du type <code>unsigned long</code> et est exprimé en s. Si cette donnée n'est pas précisée, la valeur par défaut est de 1s.</p> <p>Le résultat fourni n'est considéré comme correct que pour des impulsions de durée comprise entre 10s et 3min.</p>

6.8 FONCTIONS DE GESTION DU PORT SERIE ASYNCHRONE

La carte Arduino Uno dispose d'un port série asynchrone accessible via les E/S numériques 0 (ligne Rx) et 1 (Ligne Tx). La mémoire tampon est capable de mémoriser jusqu'à 128 caractères reçus.

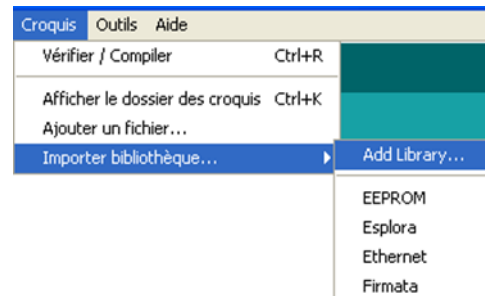
Fonctions de gestion du port série asynchrone

<code>Serial.begin(speed)</code>	Cette fonction permet de définir la vitesse de la liaison série asynchrone . Elle doit être appelée au moins une fois généralement dans la zone de la fonction <code>setup()</code>
<code>Serial.end()</code>	Cette fonction permet de désactiver la liaison série asynchrone et donc de libérer les broches numériques 0 et 1
<code>Serial.available()</code>	Cette fonction permet de connaître le nombre de caractères reçus et donc contenus dans la mémoire tampon en attente de lecture par le programme. La valeur retournée est du type <code>int</code>
<code>Serial.read()</code>	Cette fonction permet de lire le premier caractère disponible dans la mémoire tampon de réception. La valeur retournée est du type <code>int</code> (-1 si aucun caractère n'a été reçu, mémoire tampon vide)
<code>Serial.flush()</code>	Cette fonction permet de vider la mémoire tampon
<code>Serial.write(data)</code>	Cette fonction permet d' écrire sur la liaison série sous la forme binaire brute. Si la donnée est de type chaîne de caractères, les caractères sont transmis, sous le format ASCII, les uns après les autres
<code>Serial.print(data, type)</code>	Cette fonction permet d' écrire sur la liaison série en précisant le codage utilisé BIN (binaire), OCT (octal), HEX (hexadécimal), DEC (Décimal), BYTE (codage ASCII) ou une valeur comprise entre 0 et 7 pour les données à virgule flottante correspondant au nombre de décimales à transmettre

7 BIBLIOTHEQUES

Une bibliothèque est un ensemble de fonctions utilitaires mises à disposition des utilisateurs de l'environnement Arduino. Les fonctions sont regroupées en fonction de leur appartenance à un même domaine conceptuel (mathématique, graphique, tris, etc.)

L'IDE Arduino comporte par défaut plusieurs bibliothèques externes. Pour les importer dans votre programme, vous devez le menu ci-contre :



utiliser

L'instruction suivante sera alors ajoutée au début de votre programme :

```
#include <Library_name.h>
```

Bibliothèques fournies par l'IDE Arduino

EEPROM	Lecture et écriture de données dans une mémoire EEPROM
Ethernet	Connexion et transmission de données en utilisant le Shield Ethernet
Arduino Firmata	Applications utilisant un protocole série
LiquidCrystal	Contrôle d'afficheurs à cristaux liquides (LCD)
SD	Lecture et écriture de données sur des cartes SD
Servo	Contrôle de servomoteurs
SPI	Transmission de données par protocole de communication SPI (<i>Serial Peripheral Interface</i>)
SoftwareSerial	Communication série supplémentaire sur l'E/S numériques
Stepper	Commande de moteurs pas à pas
Wire	Communication TWI/I2C


D'autres librairies sont disponibles en téléchargement à l'adresse suivante :

<http://www.arduino.cc/en/Reference/Libraries>

Pour installer ces librairies, il faut décompresser le fichier téléchargé et le stocker dans un répertoire appelé *libraries* situé dans le répertoire *Arduino* créé, dans le dossier personnel, au premier lancement de l'IDE Arduino.

8 DEVELOPPEMENT D'UN PROJET

Le développement d'un projet sous Arduino nécessite les phases suivantes :

1. Lancer l'IDE Arduino en cliquant sur l'icône 
2. Éditer le programme à partir de l'IDE Arduino


```
led

/* Ce programme fait clignoter une LED branchée sur la broche n°13
 * et fait également clignoter la diode de test de la carte
 */

int ledPin = 13;          // LED connectée à la broche n°13

void setup()
{
  pinMode(ledPin, OUTPUT); // Definit la broche n°13 comme une sor
}

void loop()
{
  digitalWrite(ledPin,HIGH); // Met la sortie 13 au NL1 (diode al
  delay(3000);               // Attendre 3 s
  digitalWrite(ledPin,LOW);  // Met la sortie 13 au NL0 (diode ét
  delay(1000);               // Attendre 1 s
}
```

3. Vérifier et compiler le code à l'aide du bouton **Vérifier** 

La vérification et la compilation est correcte si aucun message d'erreur n'apparaît, le message « Compilation terminée » est notifié dans la « zone des messages des actions en cours » et aucun message d'erreur n'apparaît dans la console des « messages de compilation ».




```
Compilation terminée.

Taille binaire du croquis : 1 084 octets (d'un max de 32 256
octets)
```

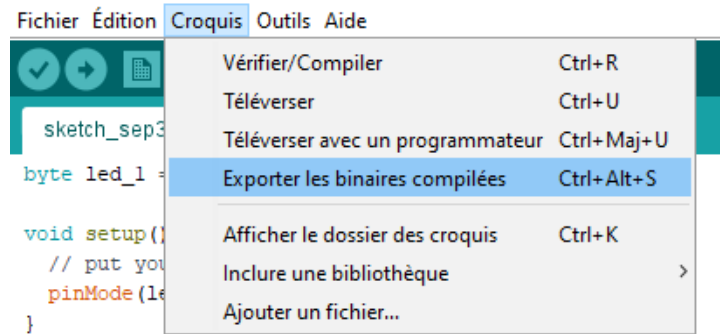
TEST SUR SIMULATEUR

4. Exporter les binaires compilées pour obtenir le programme sous forme d'un fichier binaire avec l'extension *.hex*. En réalité, le logiciel crée deux fichiers, un avec la séquence d'initialisation (*with_bootloader*) et un autre sans.

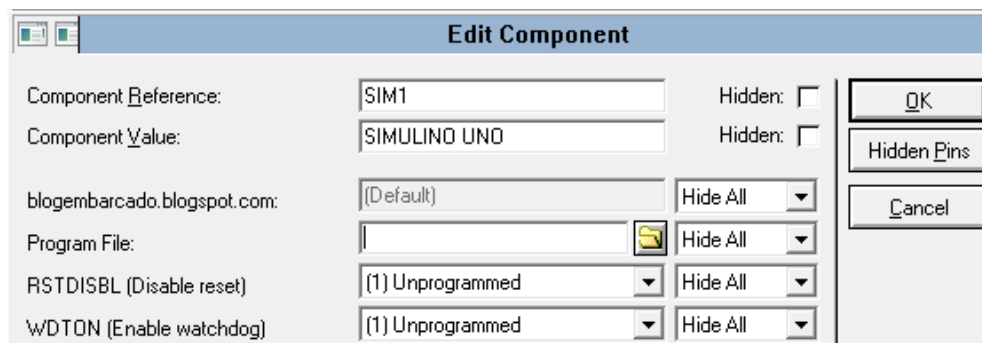
Graver la séquence d'initialisation permet d'enregistrer bootloader dans le microcontrôleur d'une carte Arduino. n'est pas requis pour l'usage normal mais peut être utile achetez un microcontrôleur vierge, livré sans bootloader.

 sketch_sep30a.ino
 sketch_sep30a.ino.standard.hex
 sketch_sep30a.ino.with_bootloader.standard.hex

un
Ceci
si vous



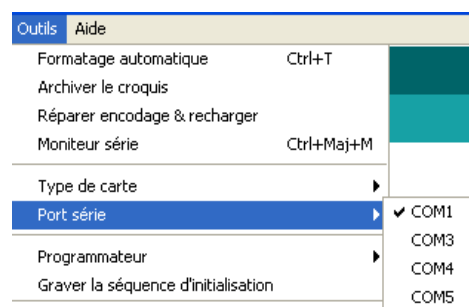
- Importer le fichier binaire `.hex` dans le logiciel de simulation **Proteus ISIS**. Éditer les propriétés de la carte Arduino pour intégrer le fichier programme.



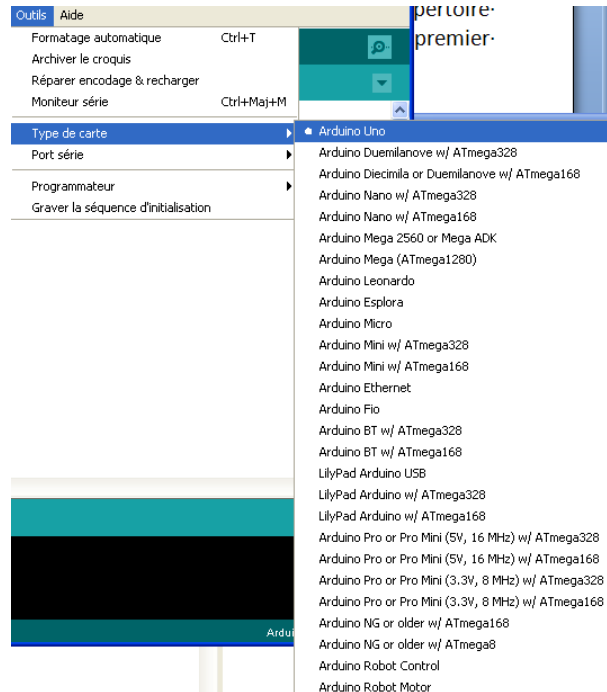
- Vérifier le fonctionnement du programme en simulant.

TEST SUR CARTE ARDUINO

- Sélectionner le bon port série à partir du menu.



8. Sélectionner la carte Arduino utilisée (Arduino Uno) à partir du menu suivant :



9. Charger le programme dans la carte Arduino en cliquant sur le bouton **Téléverser**



10. Vérifier le fonctionnement du programme sur la carte Arduino.