La boîte à outils Arduino

120 techniques pour réussir vos projets

2e édition

Michael Margolis

Traduit de l'américain
par Dominique Maniez

DUNOD

L'édition originale de ce livre a été publiée en anglais par O'Reilly Media Inc. sous le titre *Arduino Cookbook*, 2nd Edition ISBN 9781449313876.
Copyright © 2012 Michael Margolis, Nicholas Weldin.

This translation is published and sold by permission of O'Reilly Media Inc., which owns or controls all rights to publish and sell the same.

Maquette de couverture : Misteratomic

Toutes les marques citées dans cet ouvrage sont des marques déposées par leurs propriétaires respectifs.

Le pictogramme qui figure ci-contre mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit, particulièrement dans le domaine de l'édition technique et universitaire, le développement massif du photocopillage.
Le Code de la propriété intellectuelle du 1er juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établissements d'enseignement supérieur, provoquant une baisse brutale des achats de livres et de revues, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.
Nous rappelons donc que toute reproduction, partielle ou totale, de la présente publication est interdite sans autorisation de l'auteur, de son éditeur ou du Centre français d'exploitation du droit de copie (CFC, 20, rue des Grands-Augustins, 75006 Paris).

© Dunod, 2013, 2015 pour la traduction française
5 rue Laromiguière, 75005 Paris
www.dunod.com

ISBN 9782100727124

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5, 2° et 3° a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

TABLE DES MATIÈRES

© Dunod – Toute reproduction non autorisée est un délit.

© Dunod – Toute reproduction non autorisée est un délit.

© Dunod – Toute reproduction non autorisée est un délit.

AVANT-PROPOS

Cet ouvrage a été écrit avec l'aide de Nick Weldin pour vous aider à explorer toutes les choses étonnantes que vous pouvez faire avec Arduino.
Arduino est une famille de microcontrôleurs (petits ordinateurs) et un environnement de création logicielle qui facilite l'écriture de programmes (que l'on appelle *sketchs*) pouvant interagir avec le monde physique. Les choses que vous pouvez réaliser avec Arduino peuvent sentir et réagir au toucher, au son, à la position, à la chaleur et à la lumière. Ce type de technologie, que l'on désigne souvent sous le nom *d'informatique physique*, est utilisé dans toutes sortes d'appareils, de l'iPhone aux systèmes électroniques automobiles. Arduino permet à toute personne motivée (même les gens sans expérience de la programmation ni de l'électronique) d'utiliser cette technologie riche et complexe.

À qui s'adresse ce livre

Cet ouvrage s'adresse aux lecteurs intéressés par l'utilisation de la technologie informatique pour communiquer avec l'environnement. Il est conçu pour les gens qui veulent trouver rapidement la solution aux problèmes matériels et logiciels. Les recettes fournissent l'information dont vous avez besoin pour accomplir une grande variété de tâches. Il comporte aussi des détails qui vous aideront à personnaliser les solutions pour qu'elles répondent à vos besoins spécifiques. Il manque de l'espace dans un livre limité à 500 pages pour traiter la formation théorique générale, si bien que des liens vers des références externes sont indiqués tout au long de l'ouvrage (ceux qui n'ont aucune expérience en programmation ni en électronique peuvent se reporter à la section **Ce qui n'a pas été traité** pour obtenir des références générales).
Si vous avez peu d'expérience en programmation (vous avez peut-être de grandes idées sur la réalisation d'un projet interactif, mais vous n'avez pas les compétences pour le développer), ce livre va vous aider à apprendre ce que vous avez besoin de savoir pour écrire du code qui fonctionne, en utilisant des exemples qui couvrent une centaine de tâches courantes.
Si vous êtes un programmeur expérimenté, mais que vous êtes néophyte en Arduino, le livre va vous aider à devenir productif rapidement en vous montrant comment implémenter les fonctionnalités spécifiques de l'Arduino pour votre projet.
Les lecteurs qui ont déjà utilisé un Arduino devraient trouver dans ces pages un contenu qui favorise l'apprentissage rapide de nouvelles techniques, qui sont expliquées à l'aide d'exemples pratiques. Cela vous permettra d'envisager des projets

© Dunod – Toute reproduction non autorisée est un délit.

plus complexes en vous montrant comment résoudre des problèmes et utiliser des fonctionnalités qui vous sont peut-être nouvelles.
Les programmeurs C et C++ expérimentés trouveront des exemples d'utilisation des ressources bas niveau de l'AVR (interruptions, *timers*, I2C, Ethernet, etc.) afin de créer des applications qui se servent de l'environnement Arduino.

Organisation de cet ouvrage

Ce livre contient des informations qui couvrent une large palette des fonctionnalités de l'Arduino, des tâches courantes[1] aux technologies les plus avancées. Chaque technique est expliquée dans une recette qui montre comment implémenter une fonctionnalité spécifique. Vous n'avez pas besoin de lire tout le contenu de l'ouvrage à la suite. Quand une recette utilise une technique traitée dans une autre recette, il y a une référence au contenu de cette recette plutôt que de répéter les détails de cette recette en de multiples endroits.
Le chapitre 1, **Communications série**, décrit la manière de connecter l'Arduino pour qu'il communique avec votre ordinateur et d'autres appareils. La liaison série est la méthode la plus courante pour les entrées et les sorties de l'Arduino, et cette fonctionnalité est utilisée dans de nombreuses recettes de ce livre.
Le chapitre 2, **Entrées simples analogiques et numériques**, introduit toute une série de techniques pour lire des signaux numériques et analogiques.
Le chapitre 3, **Capteurs**, fournit des recettes qui expliquent comment utiliser des périphériques qui permettent à l'Arduino de sentir le toucher, le son, la position, la chaleur et la lumière.
Le chapitre 4, **Sortie visuelle**, traite du contrôle de la lumière. Les recettes étudient l'allumage d'une ou plusieurs LED, ainsi que le contrôle de l'éclat et de la couleur. Ce chapitre explique comment vous pouvez piloter des affichages à LED, et créer des motifs et des animations avec des matrices de LED. En outre, ce chapitre fournit une introduction générale aux sorties numériques et analogiques.
Le chapitre 5, **Sortie physique**, explique comment vous pouvez faire déplacer des objets en contrôlant des moteurs (servomoteurs, moteurs en courant continu, moteurs pas-à-pas) avec l'Arduino.
Le chapitre 6, **Sortie audio**, montre comment générer du son avec l'Arduino via un périphérique de sortie comme un haut-parleur. Vous apprendrez à jouer des notes et des mélodies simples, et lire des fichiers WAV et MIDI.
Le chapitre 7, **Contrôle distant d'appareils externes**, décrit des techniques qui permettent de communiquer avec presque tout appareil utilisant une forme de

1. NDE : Si vous débutez avec Arduino nous vous conseillons la lecture des ouvrages suivants :
Démarrer avec Arduino, M. Banzi, 3e édition, Tous Makers !, Dunod, 2015.
Arduino - Maîtrisez sa programmation et ses cartes d'interface, C. Tavernier, Dunod, 2011.
Arduino – Applications avancées, C. Tavernier, Dunod, 2012.

télécommande, notamment les téléviseurs, les équipements audio, les appareils photo, les portes de garage et les jouets. Il est basé sur les techniques employées dans les chapitres précédents pour connecter l'Arduino aux périphériques et aux modules.
Le chapitre 8, **Afficheurs**, étude l'interfaçage avec les afficheurs LCD en mode texte et en mode graphique. Ce chapitre montre comment connecter ces périphériques pour afficher du texte, faire défiler ou mettre en surbrillance des mots, et créer des symboles et des caractères spéciaux.
Le chapitre 9, **Heure et dates**, traite des fonctions intégrées de gestion du temps de l'Arduino et introduit de nombreuses autres techniques pour gérer les délais d'attente, la mesure du temps et les heures et les dates en pratique.
Le chapitre 10, **Communication avec I2C et SPI**, étudie les normes I2C (Inter-Integrated Circuit) et SPI (Serial Peripheral Interface). Ces standards fournissent des moyens simples pour le transfert d'informations numériques entre des capteurs et l'Arduino. Ce chapitre montre comment utiliser I2C et SPI pour connecter des périphériques courants. Il montre également comment connecter plusieurs cartes Arduino, à l'aide de I2C pour des applications multicartes.
Le chapitre 11, **Communication sans fil**, étudie la communication sans fil avec un XBee et d'autres modules sans fil. Ce chapitre fournit toute une série d'exemples qui vont du simple remplacement du port série par un module sans fil jusqu'aux réseaux mesh connectant plusieurs cartes à plusieurs capteurs.
Le chapitre 12, **Ethernet et mise en réseau**, décrit les nombreux moyens qui permettent à Arduino d'utiliser Internet. Vous y trouverez des exemples montrant comment employer les protocoles de communication les plus courants avec Arduino.
Les bibliothèques logicielles de l'Arduino sont un moyen standard d'ajouter des fonctionnalités à l'environnement Arduino. Le chapitre 13, **Utilisation, modification et création de bibliothèques**, explique comment employer et modifier les bibliothèques logicielles. Il fournit aussi des conseils sur la manière de créer vos propres bibliothèques.
Le chapitre 14, **Codage avancé et gestion de la mémoire**, étudie des techniques de programmation avancée ; les sujets abordés dans ce chapitre sont plus techniques que les autres recettes de ce livre car ils couvrent des choses qui sont habituellement masquées par l'emballage convivial de l'Arduino. Les techniques de ce chapitre peuvent être utilisées pour rendre vos sketchs plus efficaces (en améliorant les performances et en réduisant la taille du code de vos sketchs).
Le chapitre 15, **Utilisation de la puce du contrôleur**, montre comment accéder aux fonctions matérielles qui ne sont pas totalement présentées dans la documentation officielle du langage Arduino. Ce chapitre traite de l'utilisation bas niveau des registres d'entrée/sortie, des timers et des interruptions.
Les annexes :

– **A Composants électroniques** fournit une vue d'ensemble des composants utilisés dans cet ouvrage.

© Dunod – Toute reproduction non autorisée est un délit.

- **B Utilisation des schémas de montage et des notices techniques** explique comment utiliser les schémas et les notices techniques.
- **C Montage et connexion des circuits** fournit une brève introduction à l'utilisation des platines d'essai, à la connexion et à l'utilisation des alimentations externes et des piles, et à l'emploi de condensateurs de découplage.
- **D Astuces pour le débogage des logiciels** fournit des conseils sur le dépannage des problèmes de compilation et d'exécution des sketchs.
- **E Astuces pour le dépannage des problèmes matériels** couvre les problèmes avec les circuits électroniques.
- **F Broches analogiques et numériques** fournit des tableaux qui listent les fonctionnalités offertes par les broches des cartes standard Arduino.

Ce qui n'a pas été traité

Il n'y a pas assez de place dans ce livre pour traiter la théorie et la pratique de l'électronique, bien que des conseils soient fournis pour monter les circuits employés dans les recettes. Pour des informations détaillées, les lecteurs pourront se reporter aux documents qui sont largement disponibles sur Internet ou aux ouvrages suivants :

- *L'électronique en pratique* de Charles Platt (Eyrolles).
- *Getting Started in Electronics* de Forrest M. Mims III (Master Publishing).
- *Physical Computing* de Dan O'Sullivan et Tom Igoe (Cengage).
- *Practical Electronics for Inventors* de Paul Scherz (McGraw-Hill).

Ce livre de recettes explique comment écrire du code pour accomplir des tâches spécifiques, mais il ne s'agit pas d'une introduction à la programmation. Si vous voulez apprendre la programmation, il est souhaitable de se procurer l'un des ouvrages suivants :

- *Practical C Programming* de Steve Oualline (O'Reilly).
- *A Book on C* de Al Kelley et Ira Pohl (Addison-Wesley).

Mon livre préféré, bien qu'il ne soit pas vraiment un ouvrage pour débutants, est celui qui m'a servi à apprendre le C :

- *Le langage C* de Brian W. Kernighan et Dennis M. Ritchie (version française publiée par Dunod).

Style du code

Le code utilisé dans cet ouvrage a été adapté pour illustrer clairement le sujet traité dans chaque recette. Cela a pour conséquence que certains raccourcis courants de codage ont été évités, particulièrement dans les premiers chapitres. Les programmeurs C expérimentés utilisent souvent des expressions riches, mais condensées qui ont l'avantage de l'efficacité et l'inconvénient d'être peu lisibles pour les débutants. Par exemple, l'incrémentation des variables dans les premiers chapitres se fait par l'emploi d'expressions explicites qui sont faciles à comprendre pour les débutants en programmation :

```
result = result + 1; // incrémente le compteur
```

Nous n'utilisons donc pas le code suivant dont se servent habituellement les programmeurs expérimentés pour réaliser la même chose :

```
result++;  // incrémente à l'aide de l'opérateur de post incrémentation
```

Vous êtes libre d'employer le style que vous préférez. Les débutants seront rassurés en apprenant qu'il n'y a aucune amélioration des performances ni réduction de la taille du code en utilisant la version condensée.
Certaines expressions sont cependant si courantes qu'elles sont utilisées dans leur forme condensée. Par exemple, les expressions de boucle sont écrites de la manière suivante :

```
for(int i=0; i < 4; i++)
```

qui est équivalente à l'expression suivante :

```
int i;
for(i=0; i < 4; i = i+1)
```

Une bonne pratique en programmation consiste à s'assurer que les valeurs que l'on utilise sont valides en les vérifiant avant leur emploi dans des calculs. Cependant, afin de se concentrer sur le sujet traité dans les recettes, très peu de code de traitement d'erreur a été inclus dans les recettes.

Version de l'Arduino traitée dans cet ouvrage

Tout le code a été testé avec la version 1.0 d'Arduino.
Si vous avez des problèmes pour faire fonctionner les exemples de code, consultez l'annexe D qui traite du dépannage des problèmes logiciels. Le forum Arduino[1] est aussi un bon endroit pour poster une question si vous avez besoin d'assistance :

http://www.arduino.cc

Conventions utilisées dans cet ouvrage

Les conventions typographiques suivantes sont utilisées dans ce livre :

- *Italique* – Indique les noms de chemins, les noms de fichiers, les noms des programmes, les adresses Internet (noms de domaine et URL), ainsi que les nouveaux éléments quand ils sont définis.
- `Espacement fixe` – Indique les lignes de commande et les options qui doivent être saisies telles quelles, les noms et les mots-clés des programmes (y compris les noms des méthodes, les noms des variables et les noms des classes) et les balises HTML.

1. En français à l'adresse suivante : *http://arduino.cc/fr/*.

© Dunod – Toute reproduction non autorisée est un délit.

- **Police en gras** – Indique les lignes de code sur lesquelles l'attention est attirée.
- *Espacement fixe italique* – Indique le texte qui doit être remplacé par des valeurs fournies par l'utilisateur.

> **Note**
> Cet encart indique un conseil, une suggestion ou une note.

> **Avertissement**
> Cet encart indique un avertissement ou une mise en garde.

À propos de l'édition française

Cet ouvrage est la traduction partielle du *best-seller* sur Arduino aux États-Unis. Il s'agit du *Arduino Cookbook* (2e édition) paru chez O'Reilly (*http://shop.oreilly.com/product/0636920022244.do*).

L'ouvrage original compte plus de 700 pages et plus de 200 « recettes ». Nous en avons sélectionné 120 parmi les plus originales et les plus performantes en partant du principe qu'il existait déjà plusieurs ouvrages pour débutants en français, et que les amateurs d'Arduino avaient maintenant besoin d'ouvrages plus « avancés » pour progresser.

Le choix des 120 recettes de ce livre en français s'est fait avec l'aide précieuse de M. Christian Tavernier, auteur notamment chez Dunod (*www.dunod.com*) de deux ouvrages sur Arduino d'accès plus facile (*http://www.dunod.com/search?search=tavernier+arduino*).

Les compléments en ligne

Ils sont disponibles à l'adresse suivante :
http://www.dunod.com/contenus-complementaires/9782100727124

Vous pouvez télécharger **tout le code** de l'ouvrage. Vous remarquerez toutefois que les commentaires sont en français dans l'ouvrage papier pour plus de clarté dans les explications, mais que le code à télécharger est le code original, non traduit, tel qu'il a été rédigé par Michael Margolis.

Un accès aux pages consacrées au livre original sur le site (en anglais) de O'Reilly vous permettra de prendre connaissance de **commentaires** et d'éventuelles **mises à jour** au cours du temps.

Une **table de correspondances** entre les recettes du livre en anglais et celles choisies pour cette version française vous permettra de vous repérer dans le code des recettes.

Vous trouverez également dans ces compléments l'ensemble des **hyperliens actifs** de manière à vous éviter de les ressaisir dans votre navigateur.

1 • COMMUNICATIONS SÉRIE

1.0 Introduction

Les communications série offrent une façon simple et souple de faire communiquer l'Arduino avec un ordinateur ou d'autres périphériques. Ce chapitre explique comment envoyer et recevoir des informations grâce à cette fonctionnalité.
En connectant le port série de l'Arduino à votre ordinateur, vous pouvez charger des sketchs. Ce processus envoie des données depuis votre ordinateur vers l'Arduino et l'Arduino envoie à son tour des messages d'état à l'ordinateur pour confirmer le bon déroulement du transfert. Les recettes de ce chapitre montrent comment on peut utiliser cette liaison pour envoyer et recevoir des informations entre l'Arduino et votre ordinateur ou tout autre périphérique série.

Avertissement

Les communications série sont aussi un outil pratique pour le débogage. Vous pouvez envoyer des messages de débogage à partir de l'Arduino vers l'ordinateur et les afficher sur l'écran de l'ordinateur ou sur un affichage LCD externe.

L'Environnement de développement intégré (EDI) Arduino fournit un Moniteur série (illustré à la figure 1.1) pour afficher les données série envoyées à partir de l'Arduino.

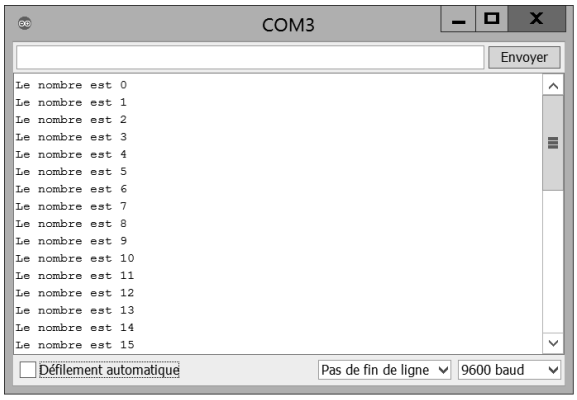

Figure 1.1 — Écran du Moniteur série de l'Arduino.

Vous pouvez aussi envoyer des données à partir du Moniteur série vers l'Arduino en saisissant du texte dans la zone située à gauche du bouton Envoyer. La valeur

© Dunod – Toute reproduction non autorisée est un délit.

en bauds (la vitesse à laquelle les données sont transmises est exprimée en bits par seconde) est sélectionnée dans la liste déroulante située en bas à droite. Vous pouvez utiliser la liste intitulée « Pas de fin de ligne » pour générer automatiquement un retour de chariot ou une combinaison de retour de chariot et de nouvelle ligne à la fin de chaque message envoyé grâce au bouton Envoyer, en choisissant l'une des options de la liste déroulante.
Un sketch Arduino peut utiliser le port série pour accéder de manière indirecte (en général via un programme proxy écrit dans un langage tel que Processing) à toutes les ressources (mémoire, écran clavier, souris, connectivité réseau, etc.) que possède votre ordinateur. Ce dernier peut aussi utiliser la liaison série pour communiquer avec des capteurs ou d'autres périphériques connectés à l'Arduino.
La mise en œuvre des communications série implique du matériel et du logiciel. Le matériel fournit la signalisation électrique entre l'Arduino et le périphérique avec lequel il communique. Le logiciel utilise le matériel pour envoyer des octets ou des bits que le matériel connecté interprète. Les bibliothèques série Arduino masquent la plus grande partie de la complexité du matériel, mais cela aide de comprendre les bases, particulièrement si vous avez besoin de détecter les erreurs dans un projet mettant en œuvre des communications série.

Matériel série

Le matériel série envoie et reçoit des données sous la forme d'impulsions électriques qui représentent les bits séquentiels. Les zéros et les uns qui transportent les informations qui composent un octet peuvent être représentés de différentes manières. Le modèle utilisé par Arduino est 0 volt pour représenter une valeur de bit égale à 0, et 5 volts (ou 3,3 volts) pour représenter une valeur de bit égale à 1.

> **Note**
> L'utilisation de 0 volt (pour 0) et 5 volts (pour 1) est très courante. On parle alors de *niveau TTL* car c'était la manière dont les signaux étaient représentés dans l'une des premières implémentations de la logique numérique, appelée *Transistor-Transistor Logic* (TTL).

Les cartes comme le Uno, le Duemilanove, le Diecimila, le Nano et le Mega ont un circuit pour convertir le port série matériel de la puce Arduino en une connexion USB pour la liaison au port série matériel. D'autres cartes, comme le Mini, le Pro, le Pro Mini, le Boarduino, le Sanguino, et la Modern Device Bare Bones Board ne prennent pas en charge l'USB et nécessitent un adaptateur pour se connecter à un ordinateur et convertir les signaux TTL en USB. Consultez *http://www.arduino.cc/en/Main/Hardware* pour de plus amples informations sur ces cartes.
Voici une liste d'adaptateurs populaires :

– Adaptateur Mini USB (*http://arduino.cc/en/Main/MiniUSB*)

– Adaptateur USB Serial Light (*http://arduino.cc/en/Main/USBSerial*)
– Adaptateur FTDI USB TTL (*http://www.ftdichip.com/Products/FT232R.htm*)
– Carte Modern Device USB BUB (*http://shop.moderndevice.com/products/usb-bub*)
– Seeedstudio UartSBee (*http://www.seeedstudio.com/depot/uartsbee-v31-p-688.html*)

Certains périphériques série utilisent le standard RS-232 pour la connexion série. Ils ont en général un connecteur à neuf broches, et un adaptateur est nécessaire pour les utiliser avec l'Arduino. La norme RS-232 est un protocole de communication ancien qui utilise des niveaux de tension incompatibles avec les broches numériques de l'Arduino.
On peut acheter des cartes Arduino compatibles avec les niveaux des signaux RS-232, comme le Freeduino Serial v2.0 (*http://www.nkcelectronics.com/freeduino-serial-v20-board-kit-arduino-diecimila-compatib20.html*).
Les adaptateurs RS-232 suivants connectent les signaux RS-232 aux broches 5V (ou 3,3V) de l'Arduino :

– Adaptateur RS-232 vers TTL 3V–5,5V (*http://www.nkcelectronics.com/rs232-to-ttl-converter-board-33v232335.html*)
– Kits d'adaptateurs P4 RS232 vers TTL Serial (*http://shop.moderndevice.com/products/p4*)
– RS232 Shifter SMD (*http://www.sparkfun.com/commerce/product_info.php?products_id=449*)

Un Arduino standard a un seul port série matériel, mais la communication série est aussi possible en utilisant des bibliothèques logicielles pour émuler des ports supplémentaires (canaux de communication) afin de fournir la connectivité à plusieurs périphériques. La communication série logicielle nécessite beaucoup de ressources de la part du contrôleur Arduino pour envoyer et recevoir des données, si bien qu'elle n'est pas aussi rapide et efficace que la communication série matérielle. L'Arduino Mega possède quatre ports matériels série qui peuvent communiquer avec quatre périphériques série différents. Un seul a un adaptateur USB intégré (vous pouvez câbler un adaptateur USB-TTL sur les autres ports série). Le tableau 1.1 liste les noms des ports et les broches utilisées par les ports série du Mega.

Tableau 1.1 — Ports série de l'Arduino Mega

Nom du port	Broche de transmission	Broche de réception
Serial	1 (aussi USB)	0 (aussi USB)
Serial1	18	19
Serial2	16	17
Serial3	14	15

© Dunod – Toute reproduction non autorisée est un délit.

Communication série logicielle

On utilise en général la bibliothèque série Arduino intégrée pour communiquer avec les ports série matériels. Les bibliothèques série simplifient l'utilisation des ports série en masquant la complexité du matériel.
Il arrive parfois que l'on ait besoin de plus de ports série que le nombre de ports matériels disponibles. Si tel est le cas, vous pouvez utiliser une bibliothèque supplémentaire qui émule par programme le matériel série. Les recettes 1.13 et 1.14 montrent comment utiliser une bibliothèque série logicielle pour communiquer avec plusieurs périphériques.

Protocole de message série

Les bibliothèques série matérielles ou logicielles gèrent l'envoi et la réception des informations. Ces informations sont souvent composées de groupes de variables qui ont besoin d'être envoyées ensemble. Pour que ces informations soient interprétées correctement, le destinataire doit reconnaître le début et la fin de chaque message. Une communication série intelligible, ou toute forme de communication impliquant des machines, ne peut s'accomplir que si l'émetteur et le destinataire s'accordent sur la manière dont les informations sont organisées dans le message. L'organisation formelle d'un message et l'étendue des réponses appropriées aux demandes s'appelle un *protocole de communication*.
Les messages peuvent contenir un ou plusieurs caractères spéciaux qui identifient le début du message — que l'on appelle *en-tête*. Un ou plusieurs caractères peuvent aussi être utilisés pour identifier la fin du message — que l'on appelle *terminateur*. Les recettes de ce chapitre illustrent des exemples de messages dans lesquels les valeurs qui composent le corps du message peuvent être envoyées soit au format texte, soit au format binaire.
L'envoi et la réception de messages au format texte impliquent l'envoi de commandes et de valeurs numériques sous la forme de lettres et de mots lisibles par un être humain. Les nombres sont envoyés sous la forme de chaînes de chiffres qui représentent la valeur. Par exemple, si la valeur est 1234, les caractères 1, 2, 3 puis 4 sont envoyés sous la forme de caractères individuels.
Les messages binaires se composent des octets que l'ordinateur utilise pour représenter les valeurs. Les données binaires sont en général plus efficaces (car elles nécessitent moins d'octets à envoyer), mais les données ne sont pas aussi lisibles que du texte, ce qui les rend plus difficiles à déboguer. Par exemple, Arduino représente la valeur 1234 par les octets 4 et 210 (4 * 256 + 210 = 1234). Si le périphérique auquel vous êtes connecté n'envoie ou ne reçoit que des données binaires, c'est le format que vous aurez à utiliser, mais si vous avez le choix, les messages au format texte sont plus simples à implémenter et à déboguer.

Il y a de nombreuses manières d'aborder les problèmes logiciels et certaines recettes de ce chapitre illustrent deux ou trois façons différentes d'arriver au même résultat. Les différences (par exemple, l'envoi de données au format texte plutôt que binaire) permettent d'offrir un équilibre entre la simplicité et l'efficacité. Quand vous avez le choix, préférez la solution que vous trouvez la plus facile à comprendre et à adapter — ce sera probablement la première solution traitée. Les solutions alternatives peuvent être un peu plus efficaces ou plus appropriées pour un protocole spécifique, mais la « bonne méthode » est celle que vous trouvez la plus simple pour votre projet.

Environnement de développement Processing

Certains exemples de ce chapitre utilisent le langage Processing pour envoyer et recevoir des messages série sur un ordinateur communiquant avec un Arduino.

Processing est un outil gratuit open source qui emploie un environnement de développement similaire à Arduino, mais au lieu d'exécuter vos sketchs sur un microcontrôleur, vos sketchs Processing s'exécutent sur votre ordinateur. Vous en apprendrez plus sur Processing et téléchargerez tout ce dont vous avez besoin en visitant le site *http://processing.org*.

Processing est basé sur le langage Java, mais les exemples de code Processing de cet ouvrage devraient être faciles à traduire dans d'autres environnements qui prennent en charge les communications série. Processing est livré avec des exemples de sketchs qui illustrent la communication entre Arduino et Processing. SimpleRead est un exemple Processing qui inclut du code Arduino. Dans Processing, sélectionnez File→Examples→Libraries→Serial→SimpleRead pour voir un exemple qui lit des données à partir du port série et modifie la couleur d'un rectangle quand on appuie sur un interrupteur connecté à un Arduino puis qu'on le relâche.

Nouveautés de l'Arduino 1.0

Arduino 1.0 a introduit un certain nombre d'améliorations et de modifications des fonctions Serial :

- `Serial.flush` attend à présent que toutes les données sortantes soient envoyées au lieu d'annuler les données reçues. Vous pouvez utiliser l'instruction suivante pour annuler toutes les données du buffer de réception : `while(Serial.read() >= 0) ; // vide le buffer de réception`
- `Serial.write` et `Serial.print` ne sont pas bloquantes. Le code précédent attendait que tous les caractères soient envoyés avant de rendre la main. À partir de la version 1.0, les caractères envoyés avec `Serial.write` sont transmis en arrière-plan (à partir d'un gestionnaire d'interruption), ce qui permet au code du sketch de retourner immédiatement au traitement. C'est en général une bonne chose (cela rend le sketch plus réactif), mais on souhaite parfois attendre que tous les caractères soient envoyés. On peut réaliser cela en appelant `Serial.flush()` immédiatement après `Serial.write()`.
- Les fonctions d'affichage Serial retournent le nombre de caractères affichés. Cela est utile quand le texte de la sortie a besoin d'être aligné ou pour les applications qui envoient des données incluant le nombre total de caractères envoyés.

© Dunod – Toute reproduction non autorisée est un délit.

– Il y a une fonctionnalité d'analyse intégrée pour les flux comme Serial afin d'extraire facilement les nombres et trouver du texte. Reportez-vous à la discussion de la recette 1.5 pour en apprendre plus sur l'utilisation de cette fonctionnalité.
– La bibliothèque SoftwareSerial qui est livrée avec Arduino a été beaucoup améliorée (voir les recettes 1.13 et 1.14).
– Une fonction `Serial.peek` a été ajoutée pour permettre de lire le caractère suivant du buffer de réception. À la différence de `Serial.read`, le caractère n'est pas supprimé du buffer avec `Serial.peek`.

Compléments

Un tutoriel sur le protocole RS-232 est disponible à *http://www.arduino.cc/en/Tutorial/ArduinoSoftwareRS232*. On trouve aussi de nombreuses informations et des liens sur le site web Serial Port Central, *http://www.lvr.com/serport.htm*.
Il existe de plus des livres sur le langage Processing :

– *Getting Started with Processing* : *A Quick, Hands-on Introduction* de Casey Reas et Ben Fry (Make).
– *Processing: A Programming Handbook for Visual Designers and Artists* de Casey Reas et Ben Fry (MIT Press).
– *Visualizing Data* de Ben Fry (O'Reilly).
– *Processing : Creative Coding and Computational Art* de Ira Greenberg (Apress).
– *Making Things Talk* de Tom Igoe (Make). Cet ouvrage qui couvre Processing et Arduino fournit de nombreux exemples de code de communication.

1.1 Envoi des informations de débogage de l'Arduino à l'ordinateur

Problème

On veut envoyer du texte et des données pour les afficher sur l'ordinateur en utilisant l'EDI Arduino ou le programme de terminal série de votre choix.

Solution

Ce sketch affiche des nombres séquentiels sur le Moniteur série :

```
/*
* Sketch SerialOutput
* Affiche des nombres sur le port série
*/
void setup()
{
  Serial.begin(9600); // envoie et reçoit à 9600 bauds
```

```
}
int number = 0;
void loop()
{
  Serial.print("Le nombre est ");
  Serial.println(number);    // affiche le nombre
  delay(500); // on attend une demi-seconde entre les nombres
  number++;  // on passe au nombre suivant
}
```

Connectez l'Arduino à votre ordinateur et chargez ce sketch. Cliquez sur l'icône du Moniteur série dans l'EDI et vous devriez voir apparaître la sortie suivante :

```
Le nombre est 0
Le nombre est 1
Le nombre est 2
```

Discussion

Pour afficher du texte et des nombres à partir d'un sketch sur un ordinateur par le biais d'une liaison série, placez l'instruction `Serial.begin(9600)` dans `setup()`, puis utilisez les commandes `Serial.print()` pour afficher le texte et les valeurs que vous voulez voir apparaître.

Le Moniteur série peut afficher des données série envoyées à partir de l'Arduino. Pour démarrer le Moniteur série, cliquez sur l'icône du Moniteur série dans la barre d'outils qui est illustrée à la figure 1.2. Une nouvelle fenêtre s'ouvre et affiche la sortie de l'Arduino.

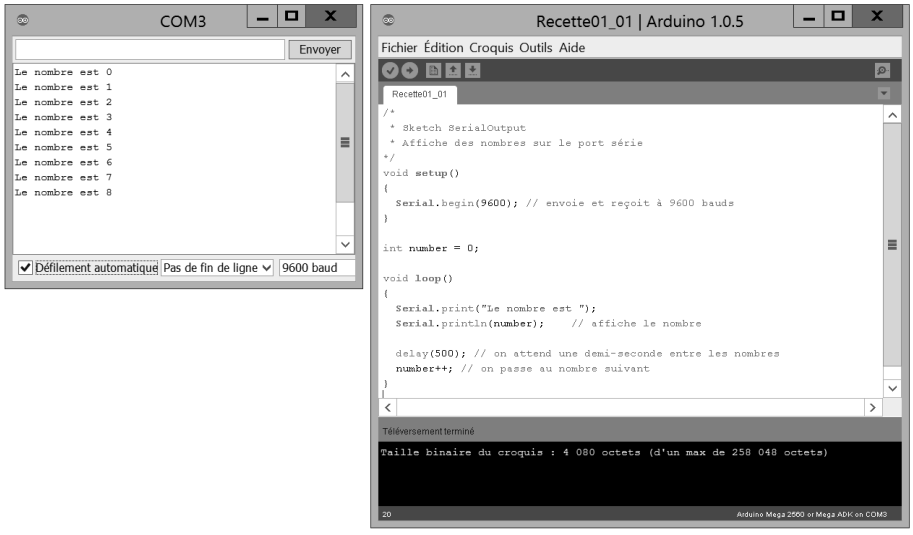

Figure 1.2 — Écran du Moniteur série Arduino.

© Dunod – Toute reproduction non autorisée est un délit.

Votre sketch doit appeler la fonction `Serial.begin()` avant de pouvoir utiliser l'entrée ou la sortie série. La fonction n'a qu'un seul paramètre : la vitesse de communication souhaitée. Vous devez utiliser la même vitesse pour l'envoi et la réception, sinon vous verrez apparaître n'importe quoi à l'écran (voire rien du tout). Cet exemple, ainsi que la plupart des autres exemples de ce livre, utilise une vitesse de 9 600 bauds (le *baud* est la mesure du nombre de bits transmis en une seconde). La vitesse de 9 600 bauds correspond approximativement à 1 000 caractères par seconde. Vous pouvez utiliser des vitesses plus lentes ou plus rapides (allant de 300 à 115 200 bauds), mais il faut vous assurer que l'envoi et la réception utilisent la même vitesse. Le Moniteur série définit la vitesse à l'aide d'une liste déroulante (en bas à droite de la fenêtre du Moniteur série ; voir la figure 1.2). Si votre sortie ressemble à ceci :

```
'3??f<Ìxì□□□ü'³??f<
```

vous devez vérifier que la vitesse sélectionnée dans le Moniteur série correspond à la vitesse définie par `Serial.begin()` dans le sketch.

Note

Si les vitesses d'envoi et de réception sont définies correctement et que vous obtenez toujours des caractères ésotériques, vérifiez que vous avez sélectionné la bonne carte dans l'EDI (menu Outils→Type de carte). Toutes les cartes n'ont pas la même vitesse et si vous avez sélectionné la mauvaise, choisissez la bonne et rechargez le sketch dans la carte.

Vous pouvez afficher du texte avec la fonction `Serial.print()`. Les chaînes (texte encadré par des guillemets) seront affichées telles quelles (mais sans les guillemets). Par exemple, le code suivant :

```
Serial.print("Le nombre est ");
```

affiche :

```
Le nombre est
```

Les valeurs (nombres) que l'on affiche dépendent du type de variable (voir la recette 1. 2). Par exemple, l'affichage d'un entier va faire apparaître sa valeur numérique ; si la variable `number` est égale à `1`, le code suivant :

```
Serial.println(number);
```

va afficher :

```
1
```

Dans le sketch de cette recette, le nombre affiché est `0` au départ de la boucle, puis il est incrémenté à chaque tour de boucle. Le `ln` à la fin de `println` provoque l'affichage de l'instruction suivante sur une nouvelle ligne.
Cela devrait vous donner des idées pour afficher du texte et la valeur décimale des entiers. Reportez-vous à la recette 1.2 pour de plus amples informations sur les options de mise en forme de l'affichage.
Il peut être souhaitable d'envisager l'utilisation d'un autre programme de terminal qui possède plus de fonctionnalités que le Moniteur série, notamment l'affichage des données au format texte ou binaire (ou les deux à la fois), l'affichage des caractères de contrôle, ou la journalisation dans un fichier. Il existe de nombreux programmes de terminal tiers, dont les suivants qui sont recommandés par les utilisateurs d'Arduino :

- **CoolTerm** (*http://freeware.the-meiers.org/*) – Programme de terminal freeware convivial pour Windows, Mac et Linux.
- **CuteCom** (*http://cutecom.sourceforge.net/*) – Programme de terminal open source pour Linux.
- **Bray Terminal** (*https://sites.google.com/site/terminalbpp/*) – Exécutable gratuit pour PC.
- **GNU screen** (*http://www.gnu.org/software/screen/*) – Programme de gestion d'écran virtuel open source qui prend en charge les communications série ; livré avec Linux et Mac OS X.
- **moserial** (*http://live.gnome.org/moserial*) – Autre programme de terminal open source pour Linux.
- **PuTTY** (*http://www.chiark.greenend.org.uk/~sgtatham/putty/*) – Programme open source SSH pour Windows et Linux qui prend en charge les communications série.
- **RealTerm** (*http://realterm.sourceforge.net/*) – Programme de terminal open source pour PC.
- **ZTerm** (*http://homepage.mac.com/dalverson/zterm/*) – Shareware pour Mac.

De plus, un article du wiki Arduino explique comment configurer Linux pour qu'il communique avec un Arduino à l'aide de TTY (*http://www.arduino.cc/playground/Interfacing/LinuxTTY*).
Vous pouvez utiliser un écran LCD comme périphérique de sortie série, mais les fonctionnalités seront limitées. Vérifiez la documentation pour voir comment votre écran gère les retours de chariot car certains écrans ne génèrent pas automatiquement une nouvelle ligne après une instruction `println`.

Compléments

La bibliothèque Arduino LiquidCrystal pour l'affichage du texte sur des écrans LCD utilise des fonctionnalités proches de la bibliothèque Serial, si bien que vous

© Dunod – Toute reproduction non autorisée est un délit.

pouvez utiliser la plupart des recettes de ce chapitre avec cette bibliothèque (voir le chapitre 8).

1.2 Envoi de texte mis en forme et de données numériques à partir de l'Arduino

Problème

On veut envoyer des données série à partir de l'Arduino pour les afficher sous la forme de texte, de valeurs décimales, hexadécimales ou binaires.

Solution

On peut afficher des données sur le port série dans plusieurs formats différents ; le sketch suivant illustre toutes les options de mise en forme :

```
/*
* SerialFormatting
* Affiche des valeurs dans différents formats sur le port série
*/
char chrValue = 65;  // valeurs de départ à afficher
byte byteValue = 65;
int intValue  = 65;
float floatValue = 65.0;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.println("chrValue: ");
  Serial.println(chrValue);
  Serial.write(chrValue);
  Serial.println();
  Serial.println(chrValue,DEC);
  Serial.println("byteValue: ");
  Serial.println(byteValue);
  Serial.write(byteValue);
  Serial.println();
  Serial.println(byteValue,DEC);
  Serial.println("intValue: ");
  Serial.println(intValue);
  Serial.println(intValue,DEC);
  Serial.println(intValue,HEX);
  Serial.println(intValue,OCT);
  Serial.println(intValue,BIN);
  Serial.println("floatValue: ");
  Serial.println(floatValue);
  delay(1000); // on attend une seconde avant
  chrValue++;  // l'affichage de la valeur suivante
```

```
    byteValue++;
    intValue++;
    floatValue +=1;
  }
```

Voici un aperçu de la sortie (condensée en quelques lignes) :

```
chrValue:   A  A  65
byteValue:  65 A  65
intValue:   65 65 41 101 1000001
floatValue: 65.00
chrValue:   B  B  66
byteValue:  66 B  66
intValue:   66 66 42 102 1000010
floatValue: 66.00
```

Discussion

L'affichage d'une chaîne de caractères est simple : `Serial.print("hello world");` envoie la chaîne « hello world » sur un périphérique à l'autre bout du port série. Si vous voulez sauter à la ligne après l'affichage de la sortie, utilisez `Serial.println()` à la place de `Serial.print()`.

L'affichage de valeurs numériques peut être un peu plus compliqué. La manière dont les valeurs byte et integer sont affichées dépend du type de variable et d'un paramètre de mise en forme optionnel. Le langage est très laxiste sur la façon dont on peut faire référence à une même valeur ayant des types de données différents. Cette souplesse peut cependant être déroutante car même si les valeurs numériques sont similaires, le compilateur les considère comme ayant des types différents avec un comportement différent. Par exemple, l'affichage d'une donnée de type `char`, `byte` et `int` ayant la même valeur ne produira pas nécessairement le même résultat. Voici quelques exemples particuliers de variables qui ont toutes des valeurs identiques :

```
char asciiValue = 'A'; // En ASCII, la lettre A a la valeur 65
char chrValue = 65; // caractère signé sur 8 bits, vaut aussi 'A'
byte byteValue = 65; // caractère non signé sur 8 bits, vaut aussi 'A'
int intValue = 65; // integer signé sur 16 bits initialisé à la valeur 65
float floatValue = 65.0; // float avec une valeur de 65
```

Le tableau 1.2 illustre ce que l'on voit quand on affiche des variables avec les routines Arduino.

Note

L'expression `Serial.print(val,BYTE);` n'est plus prise en charge en Arduino 1.0.
Si votre code a besoin que des variables byte se comportent comme des variables char (c'est-à-dire qu'il les affiche comme des caractères ASCII), vous devez le modifier en utilisant `Serial.write(val);`.

© Dunod – Toute reproduction non autorisée est un délit.

Tableau 1.2 — Formats d'affichage avec *Serial.print*

Type de données	print (val)	print (val,DEC)	write (val)	print (val,HEX)	print (val,OCT)	print (val,BIN)
`char`	A	65	A	41	101	1000001
`byte`	65	65	A	41	101	1000001
`int`	65	65	A	41	101	1000001
`long`	Le format d'une valeur `long` est identique à celui d'une valeur `int`					
`float`	65.00	Affichage non pris en charge pour les valeurs en virgule flottante				
`double`	65.00	Les valeurs `double` se comportent comme les valeurs `float`				

Le sketch de cette recette utilise une ligne séparée de code source pour chaque instruction print, ce qui peut rendre les instructions print complexes volumineuses. Par exemple, pour afficher la ligne suivante :

```
A 5 secondes : vitesse = 17, distance = 120
```

on code en général de la manière suivante :

```
Serial.print("A ")
Serial.print(t)
Serial.print(" secondes : vitesse = ")
Serial.print(v)
Serial.print(", distance = ")
Serial.println(d)
```

Cela fait beaucoup de lignes de code pour une seule ligne d'affichage et vous pouvez combiner le tout de la manière suivante :

```
Serial.print("A "); Serial.print(t); Serial.print(" secondes, vitesse = ");
Serial.print(v); Serial.print(", distance = ");Serial.println(d);
```

Vous pouvez aussi employer la fonctionnalité *insertion de style* du compilateur utilisé par Arduino pour mettre en forme vos instructions print. Vous pouvez tirer parti de certaines fonctionnalités avancées de C++ (syntaxe d'insertion de streaming et modèles) dont vous pouvez vous servir si vous déclarez un modèle de streaming dans votre sketch. Pour réaliser cela, le plus simple est d'inclure la bibliothèque Streaming développée par Mikal Hart. Vous pourrez en apprendre plus sur cette bibliothèque et télécharger son code sur le site web de Mikal (*http://arduiniana.org/libraries/streaming/*).
Si vous utilisez la bibliothèque Streaming, la ligne de code suivante procure le même résultat que les lignes précédentes :

```
Serial << "A " << t << " secondes, vitesse = " << v << ", distance = " << d <<
endl;
```

Compléments

Site officiel Arduino :

- *http://arduino.cc/en/Reference/HomePage* : commandes série.
- *http://www.arduino.cc/playground/Main/StreamingOutput :* affichage en streaming (insertion de style).

1.3 Réception de données série sur l'Arduino

Problème

On veut recevoir des données sur l'Arduino depuis un ordinateur ou un autre périphérique série, par exemple pour que l'Arduino réagisse à des commandes ou à des données envoyées depuis un ordinateur.

Solution

Il est facile de recevoir des valeurs de 8 bits (char et byte), car les fonctions `Serial` utilisent des valeurs de 8 bits. Ce sketch reçoit un chiffre (de 0 à 9) et fait clignoter la LED de la broche 13 à un rythme proportionnel à la valeur reçue :

```
/*
* Sketch SerialReceive
* Fait clignoter la LED à un rythme proportionnel à la valeur reçue
*/
const int ledPin = 13; // broche à laquelle la LED est connectée
int   blinkRate=0;     // rythme de clignotement stocké dans cette variable
void setup()
{
  Serial.begin(9600); // Initialise le port série pour envoyer
                      // et recevoir à 9600 bauds
  pinMode(ledPin, OUTPUT); // Définit cette broche en sortie
}
void loop()
{
  if ( Serial.available()) // Vérifie si au moins un caractère est disponible
  {
    char ch = Serial.read();
    if( isDigit(ch) ) // est-ce un chiffre compris entre 0 et 9 ?
    {
       blinkRate = (ch - '0');      // valeur ASCII convertie en chiffre
       blinkRate = blinkRate * 100; // rythme = 100ms * valeur du chiffre
    }
  }
  blink();
}
// On fait clignoter la LED au rythme déterminé par blinkRate
void blink()
{
  digitalWrite(ledPin,HIGH);
```

© Dunod – Toute reproduction non autorisée est un délit.

```
    delay(blinkRate); // l'attente dépend de la valeur de blinkRate
    digitalWrite(ledPin,LOW);
    delay(blinkRate);
  }
```

Chargez le sketch et envoyez des messages à l'aide du Moniteur série. Ouvrez le Moniteur série en cliquant sur l'icône du Moniteur (voir la recette 1.1) et saisissez un chiffre dans la zone de texte au sommet de la fenêtre Moniteur série. En cliquant sur le bouton Envoyer, vous envoyez à l'Arduino le caractère saisi dans la zone de texte. Si vous avez bien saisi un chiffre, vous devriez voir le rythme de clignotement changer.

Discussion

La conversion des caractères ASCII reçus peut se révéler problématique si vous n'êtes pas familier avec la manière dont le code ASCII représente les caractères. L'exemple suivant convertit le caractère `ch` en sa valeur numérique :

```
blinkRate = (ch - '0');   // valeur ASCII convertie en valeur numérique
```

Les caractères ASCII de « 0 » à « 9 » ont une valeur qui va 48 à 57. La conversion de « 1 » en sa valeur numérique est réalisée en soustrayant « 0 » car « 1 » a une valeur ASCII de 49, si bien que 48 (la valeur ASCII de « 0 ») doit être soustrait pour obtenir le nombre un. Par exemple, si `ch` représente le caractère 1, sa valeur ASCII est 49. L'expression `49- '0'` est identique à `49-48`. Cela est égal à 1, qui est la valeur numérique du caractère 1.
En d'autres termes, l'expression `(ch - '0')` est identique à `(ch - 48)` et convertit la valeur ASCII de la variable `ch` en une valeur numérique.
La réception de nombres à plusieurs chiffres implique l'accumulation des caractères jusqu'à ce qu'un caractère qui ne soit pas un chiffre valide soit détecté. Le code suivant utilise les mêmes fonctions `setup()` et `blink()` que dans l'exemple précédent, mais il accumule les chiffres tant qu'un caractère de nouvelle ligne n'a pas été reçu. Il utilise la valeur accumulée pour définir le rythme de clignotement.

> **Note**
> Le caractère de nouvelle ligne (valeur 10 en ASCII) peut être ajouté automatiquement chaque fois que l'on clique sur le bouton Envoyer. Le Moniteur série a une liste déroulante en bas de la fenêtre Moniteur série (voir la figure 1.1) ; il suffit de changer l'option « Pas de fin de ligne » en « Nouvelle ligne ».

Modifiez le code de la manière suivante :

```
int value;
void loop()
{
  if( Serial.available())
```

```
  {
    char ch = Serial.read();
    if( isDigit(ch) )// est-ce un chiffre compris entre 0 et 9 ?
    {
       value = (value * 10) + (ch - '0'); // oui, on accumule la valeur
    }
    else if (ch == 10)  // est-ce un caractère de nouvelle ligne ?
    {
       blinkRate = value;  // définit blinkrate avec la valeur accumulée
       Serial.println(blinkRate);
       value = 0; // réinitialise la valeur à 0 pour être prêt
                  // pour la séquence suivante de chiffres
    }
  }
  blink();
}
```

Saisissez une valeur, par exemple `123`, dans la zone de texte du Moniteur série et cliquez sur Envoyer ; le rythme de clignotement sera défini à 123 millisecondes. Chaque chiffre est converti à partir de sa valeur ASCII en valeur numérique. Comme les nombres sont en base 10, chaque chiffre successif est multiplié par 10. Par exemple, la valeur du nombre 234 est 2 * 100 + 3 * 10 + 4. Voici le code pour accomplir cela :

```
    if( isDigit(ch) ) // est-ce un chiffre compris entre 0 et 9 ?
    {
       value = (value * 10) + (ch - '0'); // oui, on accumule la valeur
    }
```

Si vous voulez gérer des nombres négatifs, votre code doit reconnaître le signe moins (`'-'`) en en-tête. Dans cet exemple, chaque valeur numérique doit être séparée par un caractère qui n'est pas un chiffre ou un signe moins :

```
int value = 0;
int sign = 1;
void loop()
{
  if( Serial.available())
  {
    char ch = Serial.read();
    if( isDigit(ch) ) // est-ce un chiffre compris entre 0 et 9 ?
       value = (value * 10) + (ch - '0'); // oui, on accumule la valeur
    else if( ch == '-')
       sign = -1;
    else // cela suppose que tout caractère n'étant pas un chiffre
         // ou un signe moins termine la valeur
    {
       value = value * sign ;  // définit la valeur avec la valeur accumulée
       Serial.println(value);
       value = 0; // réinitialise la valeur à 0 pour être prêt
                  // pour la séquence suivante de chiffres
       sign = 1;
```

© Dunod – Toute reproduction non autorisée est un délit.

```
    }
  }
}
```

Une autre approche pour convertir les chaînes de caractères représentant des nombres consiste à utiliser la fonction de conversion du langage C appelée `atoi` (pour les variables `int`) ou `atol` (pour les variables `long`). Ces fonctions au nom obscur convertissent une chaîne en entiers ou en entiers longs. Pour les utiliser, vous devez recevoir et stocker la totalité de la chaîne dans un tableau de caractères avant de pouvoir appeler la fonction de conversion.
Ce fragment de code termine la saisie des chiffres quand le caractère saisi n'est pas un chiffre (ou si le buffer est plein) :

```
const int MaxChars = 5; // une chaîne int contient jusqu'à 5 chiffres
                        // et est terminée par un 0 pour indiquer
                        // la fin de la chaîne
char strValue[MaxChars+1]; // doit être suffisamment grand pour les chiffres
                           //  et la valeur nulle de terminaison
int index = 0;         // l'indice du tableau stockant les chiffres reçus
void loop()
{
  if( Serial.available())
  {
    char ch = Serial.read();
    if( index < MaxChars && isDigit(ch) ){
      strValue[index++] = ch; // ajoute le caractère ASCII à la chaîne;
    }
    else
    {
      // ici quand le buffer est plein ou si un caractère n'est pas un chiffre
      strValue[index] = 0;        // on termine la chaîne par un 0
      blinkRate = atoi(strValue);  // on utilise atoi pour convertir la chaîne
                                   // en un int
      index = 0;
    }
  }
  blink();
  }
```

`strValue` est une chaîne de chiffres créée à partir des caractères reçus sur le port série. `atoi` (abréviation de ASCII vers integer) est une fonction qui convertit une chaîne de caractères en un entier (`atol` convertit en un entier long).
Arduino 1.0 a ajouté la fonction `serialEvent` que vous pouvez utiliser pour gérer les caractères série entrants. Si vous avez du code au sein d'une fonction `serialEvent` dans votre sketch, celui-ci sera appelé une fois à chaque passage dans la fonction `loop`. Le sketch suivant accomplit la même tâche que le premier sketch de cette recette, mais il utilise `serialEvent` pour gérer les caractères entrants :