Ecrire une bibliothèque pour Arduino

Traduit de la page : https://www.arduino.cc/en/Hacking/LibraryTutorial

Ce document explique comment créer une bibliothèque pour Arduino. Il commence par un croquis du code Morse clignotant et explique comment convertir ses fonctions en bibliothèque. Cela permet à d'autres personnes d'utiliser facilement le code que vous avez écrit et de le mettre à jour facilement à mesure que vous améliorez la bibliothèque.

Pour plus d'informations, consultez le guide de style API pour obtenir des informations sur la création d'une bonne API de style Arduino pour votre bibliothèque.

Nous commençons par un croquis qui fait du code Morse simple:

codemorse001.ino

```
broche int = 13 ;

void setup ( )
{
  pinMode ( pin , OUTPUT ) ;
} boucle

vide ( ) {   point ( ) ; point ( ) ; point ( ) ;   tiret ( ) ; tiret (
) ; tiret ( ) ;   point ( ) ; point ( ) ; point



( ) ;
  retard ( 3000 ) ;
}

void dot ( )
{
  digitalWrite ( pin , HIGH ) ;
  retard ( 250 ) ;
  digitalWrite ( broche , LOW ) ;
  retard ( 250 ) ;
}

void dash ( )
{
  digitalWrite ( pin , HIGH ) ;
  retard ( 1000 );
  digitalWrite ( broche , LOW ) ;
  retard ( 250 ) ;
}
```

Si vous exécutez ce croquis, il clignotera le code pour SOS (un appel de détresse) sur la broche 13.

L'esquisse comporte quelques parties différentes que nous devrons apporter dans notre bibliothèque.

Premièrement, bien sûr, nous avons les fonctions dot () et dash () qui clignotent.

Deuxièmement, il y a la variable ledPin que les fonctions utilisent pour déterminer la broche à utiliser.

Enfin, il y a l'appel à pinMode () qui initialise la broche en tant que sortie.

Commençons à transformer le croquis en bibliothèque!

Vous avez besoin d'au moins deux fichiers pour une bibliothèque: un fichier d'en-tête (avec l'extension .h) et le fichier source (avec l'extension .cpp). Le fichier d'en-tête a des définitions pour la bibliothèque: essentiellement une liste de tout ce qui se trouve à l'intérieur; tandis que le fichier source a le code réel. Nous appellerons notre bibliothèque “Morse”, donc notre fichier d'en-tête sera Morse.h. Jetons un coup d'œil à ce qui s'y passe. Cela peut sembler un peu étrange au début, mais cela aura plus de sens une fois que vous verrez le fichier source qui va avec.

Le cœur du fichier d'en-tête se compose d'une ligne pour chaque fonction de la bibliothèque, enveloppée dans une classe avec toutes les variables dont vous avez besoin:

classMorse.ino

```
class Morse
{
  public :
    Morse ( int pin ) ; point
    vide ( ) ; tiret
    vide ( ) ;
  privé :
    int _pin ;
} ;
```

Une classe est simplement une collection de fonctions et de variables qui sont toutes conservées ensemble au même endroit. Ces fonctions et variables peuvent être publiques , ce qui signifie qu'elles sont accessibles aux personnes utilisant votre bibliothèque, ou privées , ce qui signifie qu'elles ne sont accessibles qu'à partir de la classe elle-même. Chaque classe a une fonction spéciale appelée constructeur , qui est utilisée pour créer une instance de la classe. Le constructeur a le même nom que la classe et aucun type de retour.

Vous avez besoin de quelques autres éléments dans le fichier d'en-tête. L'une est une instruction **#include** qui vous donne accès aux types et constantes standard du langage Arduino (cela est automatiquement ajouté aux esquisses normales, mais pas aux bibliothèques). Cela ressemble à ceci (et va au-dessus de la définition de classe donnée précédemment):

```
#include "Arduino.h"


//Enfin, il est courant d'envelopper tout le fichier d'en-tête dans une
```

```
construction étrange:

#ifndef Morse_h
#define Morse_h

// l'instruction et le code #include vont ici ...

#endif
```

Fondamentalement, cela évite les problèmes si quelqu'un # inclut accidentellement votre bibliothèque deux fois.

Enfin, vous mettez généralement un commentaire en haut de la bibliothèque avec son nom, une brève description de ce qu'elle fait, qui l'a écrit, la date et la licence.

Jetons un coup d'œil au fichier d'en-tête complet:

```
/ *
  Morse.h - Bibliothèque pour faire clignoter le code Morse.
  Créé par David A. Mellis, 2 novembre 2007.
  Relâché dans le domaine public.
* /
#ifndef Morse_h
#define Morse_h

#include la

classe "Arduino.h" Morse
{
  public :
    Morse ( int pin ) ; point
    vide ( ) ; tiret
    vide ( ) ;
  privé :
    int _pin ;
} ;

#fin si
```

Passons maintenant en revue les différentes parties du fichier source, Morse.cpp.

Viennent d'abord quelques déclarations #include. Celles-ci donnent au reste du code accès aux fonctions Arduino standard et aux définitions de votre fichier d'en-tête:

```
#include "Arduino.h"
#include "Morse.h"
```

Puis vient le constructeur. Encore une fois, cela explique ce qui doit se passer lorsque quelqu'un crée une instance de votre classe. Dans ce cas, l'utilisateur spécifie la broche qu'il souhaite utiliser. Nous configurons la broche en tant que sortie, enregistrez-la dans une variable privée pour une utilisation dans les autres fonctions:

```
Morse :: Morse ( int pin )
{
  pinMode ( pin , OUTPUT ) ;
  _pin = broche ;
}
```

Il y a quelques choses étranges dans ce code. Le premier est le Morse :: avant le nom de la fonction. Cela dit que la fonction fait partie de la classe Morse . Vous le verrez à nouveau dans les autres fonctions de la classe. La deuxième chose inhabituelle est le trait de soulignement dans le nom de notre variable privée, _pin . Cette variable peut en fait avoir le nom de votre choix, à condition qu'elle corresponde à la définition du fichier d'en-tête. L'ajout d'un trait de soulignement au début du nom est une convention courante pour indiquer clairement quelles variables sont privées, et aussi pour distinguer le nom de celui de l'argument de la fonction (pin dans ce cas).

Vient ensuite le code réel du croquis que vous transformez en bibliothèque (enfin!). Cela ressemble à peu près au même, sauf avec Morse :: devant les noms des fonctions, et _pin au lieu de pin :

```
void Morse :: dot ( )
{
  digitalWrite ( _pin , HIGH ) ;
  retard ( 250 ) ;
  digitalWrite ( _pin , LOW ) ;
  retard ( 250 ) ;
}

void Morse :: dash ( )
{
  digitalWrite ( _pin , HIGH ) ;
  retard ( 1000 ) ;
  digitalWrite (_pin , FAIBLE ) ;
  retard ( 250 ) ;
}
```

Enfin, il est courant d'inclure également l'en-tête de commentaire en haut du fichier source. Voyons le tout:

```
/ *
  Morse.cpp - Bibliothèque pour faire clignoter le code Morse.
  Créé par David A. Mellis, 2 novembre 2007.
  Relâché dans le domaine public.
* /

#include "Arduino.h"
#include "Morse.h"

Morse :: Morse ( int pin )
{
  pinMode ( pin , OUTPUT ) ;
  _pin = broche ;
}
```

```
void Morse :: dot ( )
{
  digitalWrite ( _pin , HIGH ) ;
  retard ( 250) ;
  digitalWrite ( _pin , LOW ) ;
  retard ( 250 ) ;
}

void Morse :: dash ( )
{
  digitalWrite ( _pin , HIGH ) ;
  retard ( 1000 ) ;
  digitalWrite ( _pin , LOW ) ;
  retard ( 250 ) ;
}
```

Et c'est tout ce dont vous avez besoin (il y a d'autres trucs optionnels intéressants, mais nous en reparlerons plus tard). Voyons comment vous utilisez la bibliothèque.

Tout d'abord, créez un répertoire Morse dans le sous-répertoire des bibliothèques de votre répertoire de carnet de croquis. Copiez ou déplacez les fichiers Morse.h et Morse.cpp dans ce répertoire. Lancez maintenant l'environnement Arduino. Si vous ouvrez le menu Sketch> Import Library , vous devriez voir Morse à l'intérieur. La bibliothèque sera compilée avec les croquis qui l'utilisent. Si la bibliothèque ne semble pas se construire, assurez-vous que les fichiers se terminent vraiment par .cpp et .h (sans extension .pde ou .txt supplémentaire, par exemple).

Voyons comment nous pouvons répliquer notre ancienne esquisse SOS en utilisant la nouvelle bibliothèque:

```
#include <Morse.h>

Morse morse ( 13 ) ;

void setup ( )
{
}

void loop ( )
{
  morse. point ( ) ; morse. point ( ) ; morse. point ( ) ;
  morse. tiret ( ) ; morse. tiret ( ) ; morse. tiret ( ) ;
  morse. point ( ) ; morse.point ( ) ; morse. point ( ) ;
  retard ( 3000 ) ;
}
```

Il y a quelques différences avec l'ancienne esquisse (outre le fait qu'une partie du code a été déplacée vers une bibliothèque).

Tout d'abord, nous avons ajouté une instruction #include en haut de l'esquisse. Cela rend la bibliothèque Morse disponible pour l'esquisse et l'inclut dans le code envoyé au tableau. Cela signifie

que si vous n'avez plus besoin d'une bibliothèque dans une esquisse, vous devez supprimer l'instruction #include pour économiser de l'espace.

Deuxièmement, nous créons maintenant une instance de la classe Morse appelée morse :

```
Morse morse ( 13 ) ;
```

Lorsque cette ligne est exécutée (ce qui se produit en fait même avant la fonction setup ()), le constructeur de la classe Morse sera appelé et passera l'argument que vous avez donné ici (dans ce cas, seulement 13).

Notez que notre setup () est maintenant vide; c'est parce que l'appel à pinMode () se produit à l'intérieur de la bibliothèque (lorsque l'instance est construite).

Enfin, pour appeler les fonctions dot () et dash () , nous devons les préfixer avec morse. - le nom de l'instance que nous voulons utiliser. Nous pourrions avoir plusieurs instances de la classe Morse, chacune sur sa propre broche stockée dans la variable privée _pin de cette instance. En appelant une fonction sur une instance particulière, nous spécifions les variables d'instance à utiliser lors de cet appel à une fonction. Autrement dit, si nous avions les deux:

```
Morse morse ( 13 ) ;
Morse morse2 ( 12 ) ;
```

puis à l'intérieur d'un appel à morse2.dot () , _pin serait 12.

Si vous avez essayé la nouvelle esquisse, vous avez probablement remarqué que rien de notre bibliothèque n'était reconnu par l'environnement et mis en évidence en couleur. Malheureusement, le logiciel Arduino ne peut pas automatiquement déterminer ce que vous avez défini dans votre bibliothèque (même si ce serait une fonctionnalité intéressante à avoir), vous devez donc lui donner un peu d'aide. Pour ce faire, créez un fichier appelé keywords.txt dans le répertoire Morse. Ça devrait ressembler à ça:

```
Morse KEYWORD1
tiret KEYWORD2
dot KEYWORD2
```

Chaque ligne porte le nom du mot-clé, suivi d'une tabulation (pas d'espaces), suivi du type de mot-clé. Les classes doivent être KEYWORD1 et sont de couleur orange; les fonctions doivent être KEYWORD2 et seront marron. Vous devrez redémarrer l'environnement Arduino pour qu'il reconnaisse les nouveaux mots-clés.

Il est également agréable de fournir aux gens un exemple de croquis qui utilise votre bibliothèque. Pour ce faire, créez un répertoire d' exemples dans le répertoire Morse . Ensuite, déplacez ou copiez le répertoire contenant le sketch (appelons-le SOS) que nous avons écrit ci-dessus dans le répertoire des exemples. (Vous pouvez trouver l'esquisse en utilisant la commande Sketch> Show Sketch Folder .) Si vous redémarrez l'environnement Arduino (c'est la dernière fois, je vous le promets) - vous verrez un élément Library-Morse dans le menu Fichier> Sketchbook> Exemples contenant votre exemple. Vous voudrez peut-être ajouter des commentaires qui expliquent mieux comment utiliser votre bibliothèque.

Si vous souhaitez consulter la bibliothèque complète (avec mots-clés et exemple), vous pouvez la

télécharger: Morse.zip .

C'est tout pour le moment, mais j'écrirai probablement bientôt un tutoriel de bibliothèque avancé. En attendant, si vous avez des problèmes ou des suggestions, veuillez les publier sur le forum de développement de logiciels .

Pour plus d'informations, consultez le guide de style API pour obtenir des informations sur la création d'une bonne API de style Arduino pour votre bibliothèque.