Library Tutorial for ArduinoAuthor: Alexander BrevigContact: alexanderbrevig@gmail.com

How to write libraries for the Arduino?

So you want to write a Library for the Arduino?

-Great!

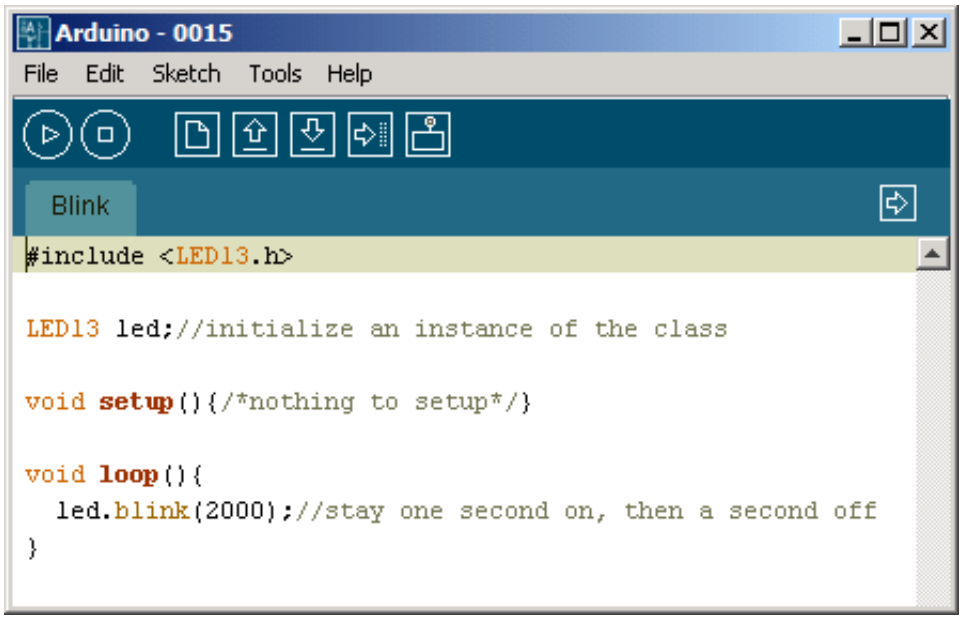
Here is a little tutorial on how to write one.

You will learn the basic syntax rules for c++ class declaration and definitic and I will summarize the conventional Arduino function and variable nami

For this tutorial we will implement a library called LED13 we will do some fun stuff with the LED @ pin 13

We will make it turn on, off and make it blink.

We will make this:



Navigation

What is a Library?

A Library is a collection of instructions that are designed to execute a spec task.
Good libraries describe the tasks it tries to solve by the name of the librar itself, and the names of the functions associated to the library.
This is because the library should be intuitive to use, and all names shoulc describe, in an efficient way, all questions regarding *what*, *when*, and *whe* but should conceal the *how*.

On the Arduino the libraries often serve as a wrapper around the Arduino API language to enhance code *readability* and *user experience*.

Arduino IDE places it's libraries in \hardware\libraries The Library for this tutorial has the URL C:\programming\arduino-0015\hardware\libraries\LED13 on my Windows XP machine.

For the Ubuntu operating system, the libraries may be found at /usr/share/arduino/libraries.

What is the Arduino conventions?

The Arduino libraries uses a UppercaseUppercase convention for Library names, and lowercaseUppercase naming convention for the functions. What this means is that our library has/*should* get the name LED13 (not led13 for instance), and the functions will be named on() and off().
I could've selected turnOn() and turnOff(), but I've selected the shortest because it is as descriptive, but in addition it is shorter. [short code == le probability for typos]

It is important to remember that you do not *need* a Library. It only make simpler.
If you are interested in the inner mechanics of a specific library, you'll finc the*MyLibrary*.cpp (if downloaded) in /hardware/libraries/*MyLibrary*/*MyLibrary*.cpp change*MyLibrary* to whatev name of the library you want to investigate

---

Why do we have libraries?

- Simplify usage and/or organization of code
- Enhance code readability
- Decentralize logic
    - Main sketch can now simply state library.action() instead of coding the equivalent in the sketch.

How do we write libraries?

[define functionality - make a proof of concept - learn c++ class syntax - implement -debug]

Define the functionality your Library has to provide

For our Library we've decided on these functions:

```
on()
off()
blink(int time)
```

This should be the premise for our library.
At this point, we do not care *how* the on() should work, only that it should exist.

Make a proof of concept sketch

Before implementing a new class I usually find it very useful to test the id in a sketch. Try a few different implementations, see what uses the least amount of space, and what seems simplest in use. Select the solution that compromises these two in a suitable way.

An experience I've made is that if one has something slightly similar to w from, that is often beneficial.
For our Library, one immediatly thinks of the Blink right? The Hello World! c µCs. So, go ahead and start the Arduino IDE and open the File->Sketchbc >Examples->Digital->Blink.
The first thing we'll do, is to cut the code inside loop(), and paste it into a new function called void blink().

LED connected to pin 13.

Now, write blink(); inside the loop, so that the code that once was inside t loop() and now is located inside blink, gets executed (because loop calls blink).

Now, you should have a sketch looking like this:

```
1.  int ledPin = 13;                // LED connected to digital pin 13
2.
3.  void setup()                    // run once, when the sketch starts
4.  {
5.    pinMode(ledPin, OUTPUT);      // sets the digital pin as output
6.  }
7.
8.  void loop()                     // run over and over again
9.  {
10.   blink(); //call the code that makes the LED blink
11. }
12.
13. //make the LED blink
14. void blink(){
15.   digitalWrite(ledPin, HIGH);   // sets the LED on
16.   delay(1000);                  // waits for a second
17.   digitalWrite(ledPin, LOW);    // sets the LED off
18.   delay(1000);                  // waits for a second
19. }
```

[Get C

The next two steps will be to add the void on(); and the void off();. These will be really simple functions, becuase they will actually just 'wrap'digitalWrite(13,HIGH) and digitalWrite(13,LOW) respectively.

Go ahead and try to write these functions.

You should end up with something like:

```
1. //turn the LED on
2. void on(){
3.   digitalWrite(ledPin,HIGH); //set the pin HIGH and thus turn LED on
4. }
5.
6. //turn the LED off
7. void off(){
8.   digitalWrite(ledPin,LOW); //set the pin LOW and thus turn LED off
9. }
```

[Get C

We have one more thing to fix.

As the code is now, the blink() will always use two seconds, but we wante to pass the number of microseconds as an argument. Additionally we'll wa to remove the redundant code that turns a led on and off. We'll replace th redundant calls with on() and off(). You probably know how to write functi that has parameters, but here it is:

```
1. //make the LED blink
2. void blink(int time){
3.   on();                    // sets the LED on
4.   delay(time/2);           // waits for a second
5.   off();                   // sets the LED off
6.   delay(time/2);           // waits for a second
7. }
```

[Get C

Now you have enough code to implement all the desired functionality

Complete Concept Sketch

```
int ledPin = 13;                // LED connected to digital pin 13

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()                     // run over and over again
{
  blink(2000);  //call the code that makes the LED blink
}

//turn the LED on
void on(){
  digitalWrite(ledPin,HIGH); //set the pin HIGH and thus turn LED on
}

//turn the LED off
void off(){
  digitalWrite(ledPin,LOW); //set the pin LOW and thus turn LED off
}

//make the LED blink
void blink(int time){
  on();                             // sets the LED on
  delay(time/2);                // waits for a second
  off();                            // sets the LED off
  delay(time/2);                // waits for a second
}
```

[Get C

Download the source for this tutorial

The next step is to:

Learn c++ class syntax

There are a lot of resources on this topic, but generally the class are split two files.

1. The declaration, referred to as the *header file*. 'LED13.h' will indicate that file declared the class LED13. *Declaration* is the process of defining what t class should do.
2. The implementation, referred to as the *source file*. 'LED13.cpp' indicates t the file implement the declared functions and variables from "LED13.h". *Implementation* is the process of writing the code, that determines how the declared functions are imlemented.

LED13 Header

```
1. #ifndef LED13_H
2. #define LED13_H
3.
4. #include <WProgram.h> //It is very important to remember this! note
   that if you are using Arduino 1.0 IDE, change "WProgram.h" to "Arduino.
5.
6. class LED13 {
7. public:
8.         LED13();
9.         ~LED13();
10.        void on();
11.        void off();
12.        void blink(int time);
13. };
```

14.
15. #endif

[Get C

'Code Analysis:'

Lines 1 and 2 are referred to as an include guard
It basically prevents the code from being included into the program binary multiple times.
Line 4 includes the WProgram code to our library. This enables us to use the pinMode, digitalWrite, delay etc.
Line 6 is the beginning of the class itself.
Line 7 uses the public keyword
Line 8 and 9 are called constructor and destructor, they are responsible for setting up your library (construct), and delete it (deconstruct)
Line 10 declares the function 'void on();' , this will turn the LED on
Line 11 declares the function 'void off();' , this will turn the LED off
Line 12 declares the function 'void blink(int time);' , this will blink the LED in 'time' milliseconc
Line 13 is the end of the class.
Line 15 ends the #ifndef preprocessor directive

LED13 Source

You will see that this source will be very much like the concept sketch.

However, I've changed some datatypes, and the rest is just the c++ synt

Notice the :: , it is called the 'scope resolution operator'

1. #include "LED13.h" //include the declaration for this class
2.
3. const byte LED_PIN = 13; //use the LED @ Arduino pin 13, this should nc change so make it const (constant)
4.
5. //<<constructor>> setup the LED, make pin 13 an OUTPUT

```
7.      pinMode(LED_PIN, OUTPUT); //make that pin an OUTPUT
8.  }
9.
10. //<<destructor>>
11. LED13::~LED13(){/*nothing to destruct*/}
12.
13. //turn the LED on
14. void LED13::on(){
15.         digitalWrite(LED_PIN,HIGH); //set the pin HIGH and thus turn LED
16. }
17.
18. //turn the LED off
19. void LED13::off(){
20.         digitalWrite(LED_PIN,LOW); //set the pin LOW and thus turn LED of
21. }
22.
23. //blink the LED in a period equal to paramterer -time.
24. void LED13::blink(int time){
25.         on();                 //turn LED on
26.         delay(time/2);  //wait half of the wanted period
27.         off();                //turn LED off
28.         delay(time/2);  //wait the last half of the wanted period
29. }
```

[Get C

Source for this tutrial

Attach:LED13.zip

Implementation

Selecting public, protected or private

A common 'rule of thumb' is to make all data as private as possible. For

functions defined public.

Selecting datatypes / returntypes

Being the Arduino has limited resources I think selecting the 'smallest' variable is the way to go. An Arduino pin will (probably) never become ab 255. So in that case using a byteis sufficient, and preferable.

Selecting variable- / functionnames

The arduino API has established some cues for how to name a variable or function. Start lower case, and indicate new words with an upper case lett 'myDescriptiveVariable'

Writing examples

Be sure to create examples that are as self describing as possible. Additionally it should be made an effort to demo most of the functions.

Blink Example

```
1. #include <LED13.h>
2.
3. LED13 led;//initialize an instance of the class
4.
5. void setup(){/*nothing to setup*/}
6.
7. void loop(){
8.   led.blink(2000);//stay one second on, then a second off
9. }
```

[Get C

keywords.txt

Remember to implement this file, as it makes using your library a lot easi

Those are:

LED13
on
off
blink

And they are implemented in keywords.txt as:

```
LED13KEYWORD1onKEYWORD2offKEYWORD2blinkKEYWORD2
```

Debugging

Remember #include <WProgram.h>

You will get a series of errors if you've forgot to include the WProgram, *and* trying to use some of the Arduino functions or datatyp It is only necessary to include the WProgram.h in you library .h, your .cpp will include it because it includes your header.

Delete object file and recompile between each alteration of header and/or source

When changes are made in the header or the source, those changes are n automatically incorporated in the object file of you library. To be sure that your changes are compiled and used, delete the LED13.o and recompile a sketch in the Arduino IDE.

NOTE: After making library changes, before recompiling, you may need to delete the <LibName>.o file which can be found by searching, or in a file similar to: c:\Documents and Settings\Administrator\Local Settings\Temp\build3998508668703718780.tmp

Learn to interpret the Arduino IDE debugger / compiler.

It can be a timesaver to take an hour and causing error deliberately, so y know what the error messages are caused by for later reference

Links

Arduino Reference

Core Libraries

Arduino Library Tutorial

Information about this page

Part of AlphaBeta Tutorials and Resources.
Last Modified: March 05, 2013, at 07:55 PM
By: Bruce_S