HELP Arduino Library Writing

Here are the steps towards writing an Arduino library. Use a plain text editor to create the files.

1. First define what the library is going to do, and decide on the functions it will provide. one way to do this is to write a sketch and get it working, without a library. Then take this sketch's functions and build up a library using them.

For example, a VFO built around a AD9850 module. This has a digital synthesiser with analog sine wave output up to 30-40MHz.

Three functions can be written to use the module

```
begin(...)          - set up the pin connections and reset the module
setFreq(...)        - make it output a frequency
calibrate(...)      - allow for an xtal slightly off frequency
```

2. The header file "`ASD9850.h`". This file contains the class name definition and the prototypes of the functions used:

```
#include <Arduino.h>

#define ADS_XTAL 125000000

class ADS9850 {

     public:
          ADS9850();

          void begin(int W_CLK, int FQ_UD, int DATA, int RESET);
          void setFreq(unsigned long Hz, float Chz);
          void calibrate(unsigned long calHz);

     private:
          int _W_CLK;
          int _FQ_UD;
          int _DATA;
          int _RESET;

          unsigned long _calFreq;

          void pulse(int _pin);

};
```

The xtal marked frequency is defined by `ADS_XTAL` as the nominal frequency, it can be changed by the `calibrate(...)` function.

The class name, which must be the same as the “.h” file name, is next, and contains the class prototype code in brackets.

The code can either be “public” or “private”. Items in the public section can be seen by users, but those in the private section are not available outside the library. So in the public sections we have the class object creation call, and the prototypes of the functions of the library.

In the public section we have

```
ADS9850(); - the call to create a synthesiser object

void begin(int W_CLK, int FQ_UD, int DATA, int RESET); // the initialise function
void setFreq(unsigned long Hz, float Chz); // the set frequency function
void calibrate(unsigned long calHz); // the calibration function
```

and in the private section we have

```
int _W_CLK;
int _FQ_UD;
int _DATA;
int _RESET;

unsigned long _calFreq;

void pulse(int _pin);
```

where you see some internal variables (which have the same name as the public variables but proceeded by an underscore. A variable used only in the private code and a prototype function also only used in the private section.

3. The basic code is enclosed in a standard envelope which looks like this

```
#ifndef ADS9850_H
#define ADS9850_H

#include <Arduino.h>

// above class definition code goes here...

#endif
```

This envelope serves two purposes, the “`ifndef, define`” commands check to see if ADS9850_H constant is defined, if not define it. Then at the end this conditional is closed. The purpose of this is to prevent the code being included in your sketch twice, should you make the mistake of writing “`#include <ADS9850.h>`” twice in your sketch...

The second part “`#include <Arduino.h>`” adds the standard Arduino libraries to your library and sketch. These are normally added to all sketches in a hidden way, but you have to include them specifically in a library.

So the complete “.h” file is

```
// Arduino Library for AD9850 Analog Digital Synthesiser module

#ifndef ADS9850_H
#define ADS9850_H

#include <Arduino.h>

#define ADS_XTAL 125000000

class ADS9850 {

     public:
          ADS9850();

          void begin(int W_CLK, int FQ_UD, int DATA, int RESET);
          void setFreq(unsigned long Hz, float Chz);
          void calibrate(unsigned long calHz);

     private:
          int _W_CLK;
          int _FQ_UD;
          int _DATA;
          int _RESET;

          unsigned long _calFreq;

          void pulse(int _pin);

};
#endif
```

3. The guts of the library is the code for the functions in a `".cpp"` (C++) file.

The first lines in this file are

```
#include "Arduino.h"
#include "ADS9850.h"

ADS9850::ADS9850() {

}
```

To include again the Arduino header files, and your own header file "ADS9850.h". Next there is a function that is called to define the synthesiser object in your code. The object's name can be anything and more than one object could be created, for example if you want to connect and drive two AD9850 modules you could initiate two objects in your sketch

```
ASD9850 synth1;
ASD9850 synth2;
```

and control them separately

```
synth1.setFreq(10000000, 0); // 10000000 Hz
synth2.setFreq(5000000, 100); // 5000001 Hz
```

Next in the C++ code are the functions, both public and private. For example

```
void ADS9850::begin(int W_CLK, int FQ_UD, int DATA, int RESET) {
     _W_CLK = W_CLK;
     _FQ_UD = FQ_UD;
     _DATA = DATA;
     _RESET = RESET;
     _calFreq = ADS_XTAL;

     pinMode(_W_CLK, OUTPUT);
     pinMode(_FQ_UD, OUTPUT);
     pinMode(_DATA, OUTPUT);
     pinMode(_RESET, OUTPUT);

     pulse(_RESET);
     pulse(_W_CLK);
     pulse(_FQ_UD);
}
```

Variables used inside the library are assigned values provided by you outside when calling the `begin(...)` function (`_W_CLK = W_CLK`, etc). It is convention to just add an underscore to

the outside name.
The Arduino pins are then set to `OUTPUT` mode. And the private function “pulse(...)” is called to reset the AD9850 with a series of `HIGH-LOW` pulses on various pins.

The other functions implemented in the CPP file are shown in the complete version below.

```
// Arduino Library for AD9850 frequency synthesiser module

#include "Arduino.h"
#include "ADS9850.h"

ADS9850::ADS9850() {

}

void ADS9850::begin(int W_CLK, int FQ_UD, int DATA, int RESET) {
     _W_CLK = W_CLK;
     _FQ_UD = FQ_UD;
     _DATA = DATA;
     _RESET = RESET;
     _calFreq = ADS_XTAL;

     pinMode(_W_CLK, OUTPUT);
     pinMode(_FQ_UD, OUTPUT);
     pinMode(_DATA, OUTPUT);
     pinMode(_RESET, OUTPUT);

     pulse(_RESET);
     pulse(_W_CLK);
     pulse(_FQ_UD);
}

void ADS9850::setFreq(unsigned long Hz, float Chz) {
     Chz /= 100.0;

     unsigned long tuneHz = Hz * pow(2, 32) / _calFreq;
     unsigned long tuneChz = Chz * pow(2, 32) / _calFreq;
     unsigned long tune = tuneHz + tuneChz;

     shiftOut(_DATA, _W_CLK, LSBFIRST, tune);
     shiftOut(_DATA, _W_CLK, LSBFIRST, tune >> 8);
     shiftOut(_DATA, _W_CLK, LSBFIRST, tune >> 16);
     shiftOut(_DATA, _W_CLK, LSBFIRST, tune >> 24);
     shiftOut(_DATA, _W_CLK, LSBFIRST, 0x00);

     pulse(_FQ_UD);
}
```

```
void ADS9850::calibrate(unsigned long calXtal) {
     _calFreq = calXtal;
}

void ADS9850::pulse(int pin) {
     digitalWrite(pin, HIGH);
     digitalWrite(pin, LOW);
}
```

With all that explanation I think you can go ahead and write your own libraries.

When you have the two files `".h"` and `",cpp"`, make a new folder with your library name, under your `"libraries"` folder and place them there. When the Arduino IDE starts it scans the libraries folder to see what is there and will find you library.

Lastly, in order that in the IDE editor your library and functions are shown in a contrasting red colour you have to make a `"keywords.txt"` file, like this

```
#######################################
# Syntax Coloring Map For ADS9850
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################

ADS9850       KEYWORD1

#######################################
# Methods and Functions (KEYWORD2)
#######################################
begin         KEYWORD2
setFreq       KEYWORD2
calibrate     KEYWORD2
#######################################
# Constants (LITERAL1)
#######################################
```

Things labeled `KEYWORD1` will be shown in red, and those labelled `KEYWORD2` in brown (you may not notice any difference on screen...)