Abstract

When a project requires the use of a microcontroller, a go to for many hobbyists and starting engineers is the Arduino/AVR platform. The community of users surrounding the Arduino/AVR platform has created a stable base over the years, contributing many parts of code, or solving problems with hardware and giving that solution back to the community. This community is an ever growing entity and with all the information that can be gathered from it, giving useful work back for others to use is exemplary. The easiest way to help others in the community is to write and present code that can be reused over and over again by others. These modules are best and most easily available in the form of Arduino Libraries.
Introduction

When working on a project that requires large amounts of code, having to start without any coding resources means that there is quite a bit of work to be done. Libraries of code make getting tasks done in future and present work much easier. Simply making a library doesn’t quite solve every problem though. There are several coding ideas to keep when making the library to be demonstrated in this document.

The project that has been presented to ECE 480 Design Team 1 while requiring some hardware design, mainly consists of writing Application Program Interfaces, meaning that large amounts of code are to be created to be reused by others. Arduino Libraries can easily be implemented and transferred between systems, making the reusability easily plausible. There are several parts that are generally required in order to create a successful Arduino Library.
Hosting

Simply creating a code library to be reused by others doesn’t give them access to that code. There are many ways to store the library conveniently so that others may have access for downloading and the creator(s) may have access for editing. There are many methods available for storing and making code easily editable, however, this note will focus only on one of those methods.

GIT

Git is a tool created for uploading code and other files to repositories.

- The repositories can be set to be publicly available or private, to be shared only with those granted access.
- When code is updated in the repository, the changes can be compared in different versions side by side.
- Once uploaded the committed code is documented and dated so that users or programmers know the last time a file was updated.
- Those with access to the code are also able to “fork” code repositories where they can edit and change without affecting the original, creating a different version.
- Branched code may be merged back together easily.
- There are GIT clients available for uploading the code directly to the repository
- The repositories allow for access to previous versions of the code.

Putting the code into a repository means that others can link to and have a source for the library created by a user. It allows for convenient storage and ability to be distributed in a simple form.
Ideas to Keep In Mind

When writing code, it is easy to slip into the idea that the programmer only needs to get something that functions, but several questions must be asked to ensure that the code produced is both productive and beneficial.

- Is the code being written application specific?
- Would making the code available publicly benefit others?
- Could the code being written be used in more than just one project?
- Is the code able to be tested in a manner that proves its reliability?
- Is the code well documented so that another could understand its usage?

Coding for projects can be extremely difficult if the end goal is only to produce something that “works.” There can be a large difference between code that works and code that is good.

With microcontrollers, there are multitudes of peripheral hardware that allows use of different features. While writing all code inline makes locating the code easier, it can cause bulky and difficult to read due to the amount of functions, methods and definitions required to successfully and meaningfully create a working program. The more objects that the microcontroller has to interface, more code for interfacing with the objects will be needed. Finding different hardware to bring together to accomplish the means of the project implies that the code written is going to be application specific. Breaking this question down, it may not make sense to publicly release this code publicly.

Making good code requires that while all the same hardware elements may not be reused on a single microcontroller, individual elements of the project may be. Writing libraries for interfacing with each device solves many of these problems from splitting up the code into smaller more sensible segments to allowing code to be reused in other areas. In library form code used to interface with another device in multiple projects without needing to be copied and pasted from one project into another. Knowing exactly what’s in the library makes testing the functions of the library much simpler, with each library for a hardware or software element able to be tested in multiple projects under different conditions.

Keeping good documentation on the code written is a must. Writing comments will help others be able to look at and understand the code written. Without comments or documentation of any form, the code will be hard to understand and will be near impossible for another person to use.

Writing code with these ideas in mind will help with both the use and reuse of the code and large amounts of saved time, later.
Parts of Arduino Library

Header and Source Files

These files are essential to writing a library for the Arduino platform, the header and source files are written in C/C++ and the library won’t have a real function without properly written files. The header contains the main definition of the library or class where the programming objects, functions/methods, and class specifiers will be defined along with comments describing the operation of functions and variables of the library class. Source files contain the “worker” code that operates on the arguments provided to function/method. These files must be titled the same as one another as well as the folder they are in.

Ex. “SomeLibrary\SomeLibrary.h”, “SomeLibrary\SomeLibrary.cpp”

Keywords

This file is used by the Arduino programming environment to highlight and bold words according to how the labeled word functions in the library itself. Once a library is included in the Arduino environment .ino file, if this file is available, it makes seeing attributes of the library easier as words related to the functions will be highlighted and bolded according to how they were labeled in the “keywords.txt” file. This file is included in the “SomeLibrary” folder as an example and would have the path “SomeLibrary\keywords.txt”

Read Me

Library names are best kept short so that there is less of a chance of making an error when including the library. When library names are kept short, sometimes getting the purpose of the library across isn’t as evident as is desired. Having a ReadMe.txt file is very helpful as it can guide a user towards the real purpose of the library. This file can also contain author, licensing and contributor information specifically related to the library as well as contact information for suggestions and bug fixing information.

Examples

This is a folder used for storing an Arduino “sketches” that demonstrate using the library created within the Arduino environment. There can be one or multiple demonstrations of using the functions/methods within the library to accomplish different tasks. Along with well written comments and a planned out example, a well written library is in the works.
Library Book Image

http://pcss.sd8.bc.ca/wp-content/uploads/2013/01/books-clipart1.jpg

Arduino Logo

https://www.stlo.unicaen.fr/sc12-g2b/images/Arduino/arduino_logo1.png

Information on GIT

http://www.git-scm.com/

Information on Arduino and Libraries

http://arduino.cc/