Arduino Libraries

Written by Mark Webster, Conner Sheeran, Ousema Zayati??

Summary
Learn to create and use libraries for the Arduino IDE development software.

Introduction

The Arduino Integrated Development Environment (IDE) is easily extended with custom libraries that supplement the libraries included with the IDE. These libraries are composed of C++ files stored in a “libraries” folder in the “Arduino” folder that contains all the users sketches (.INO programs). A library comes with an include file and usually example files.

When the user writes an *.ino sketch, they can include libraries with a line at the top of the program such as #include <SoftwareSerial.h>
The *.h file is a C include file that defines the functions, variables, classes, and methods used by the library.

Arduino libraries in the IDE are explained on the webpage
https://www.arduino.cc/en/reference/libraries

Standard Libraries

Many libraries are available by default just by installing the Arduino IDE. The do not have to be installed but usually have to be included with a line at the start of the Arduino program. For example #include <Ethernet.h>. Some default libraries must be imported using the Library Manager.

These default libraries currently are:

EEPROM - reading and writing to "permanent" storage
Ethernet - for connecting to the internet using the Arduino Ethernet Shield, Arduino Ethernet Shield 2 and Arduino Leonardo ETH
Firmata - for communicating with applications on the computer using a standard serial protocol.
GSM - for connecting to a GSM/GRPS network with the GSM shield.
LiquidCrystal - for controlling liquid crystal displays (LCDs)
SD - for reading and writing SD cards
Servo - for controlling servo motors

SPI - for communicating with devices using the Serial Peripheral Interface (SPI) Bus
SoftwareSerial - for serial communication on any digital pins. Version 1.0 and later of Arduino incorporate Mikal Hart's NewSoftSerial library as SoftwareSerial.
Stepper - for controlling stepper motors
TFT - for drawing text , images, and shapes on the Arduino TFT screen
WiFi - for connecting to the internet using the Arduino WiFi shield
Wire - Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors

There are additional libraries like “Audio.h” which only work with specific Arduino boards, like the Arduino Due.

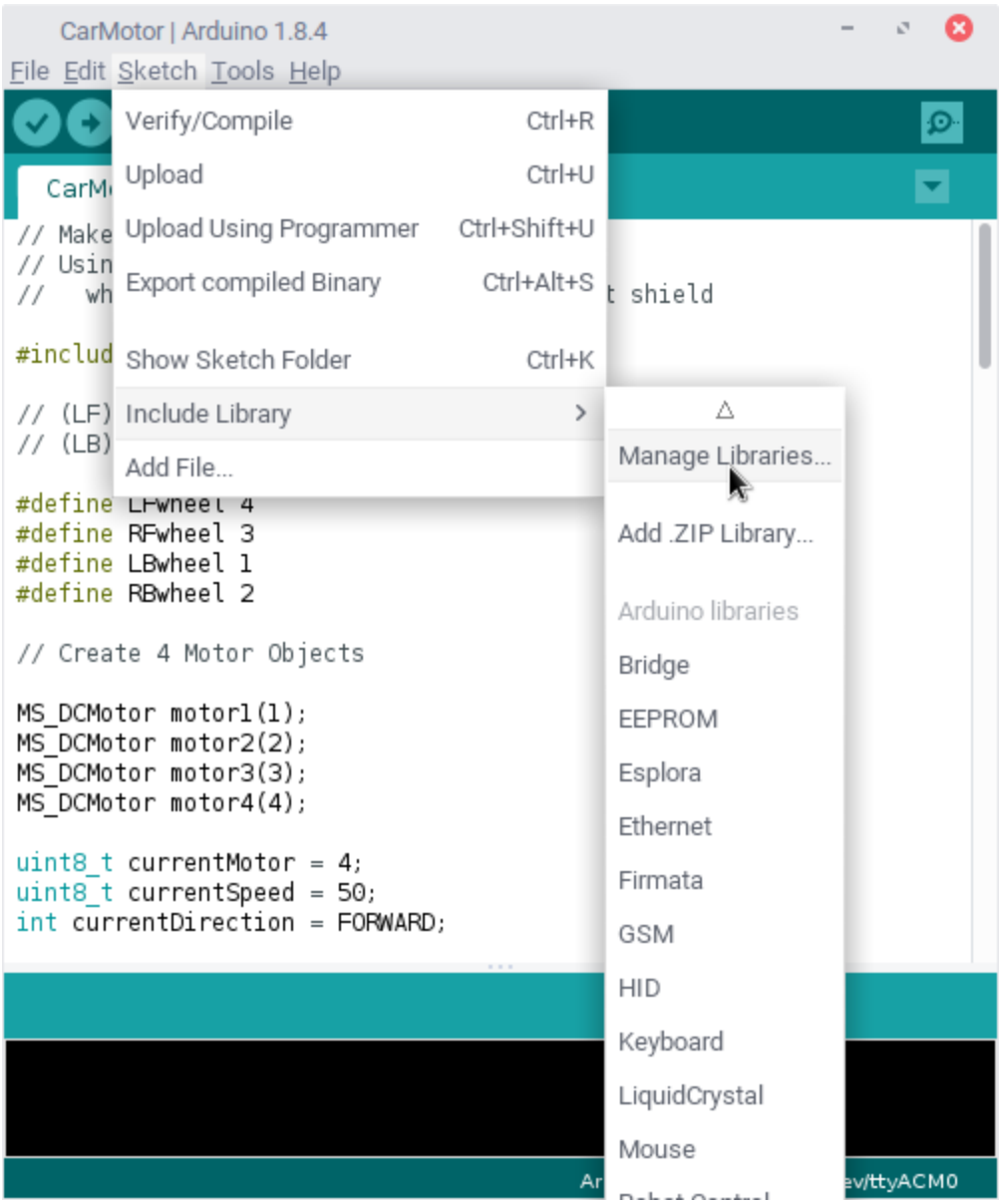
Further, there are user contributed libraries like “NewSoftSerial” which must be installed first using the guide: https://www.arduino.cc/en/Guide/Libraries

Inline Question: Where are these libraries listed in the IDE?

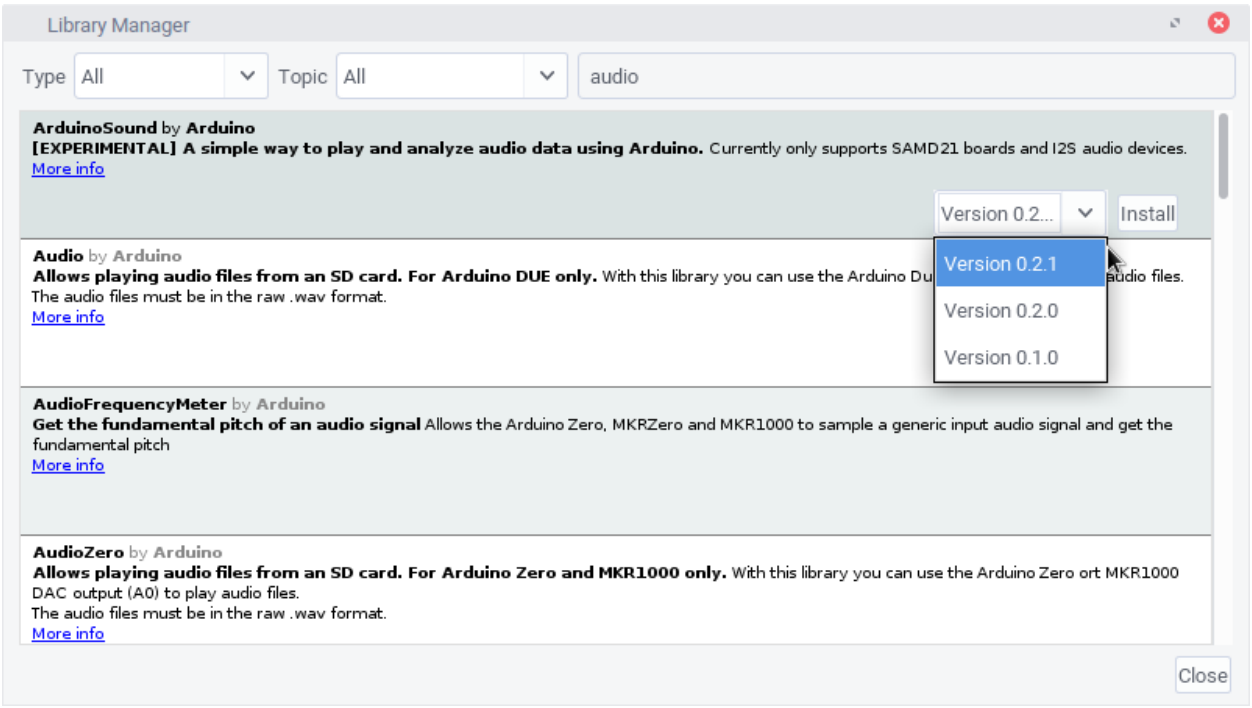
Under the menu Sketch -> Include Library

Install Using Library Manager

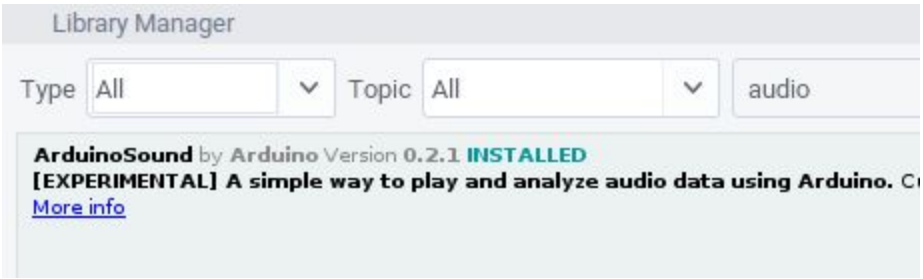
Libraries that are already on the computer with the IDE can be made available using the library manager on the Sketch menu.



The library manager has a searchable list of available libraries. If the particular version of the library isn't installed, click on the Install button.

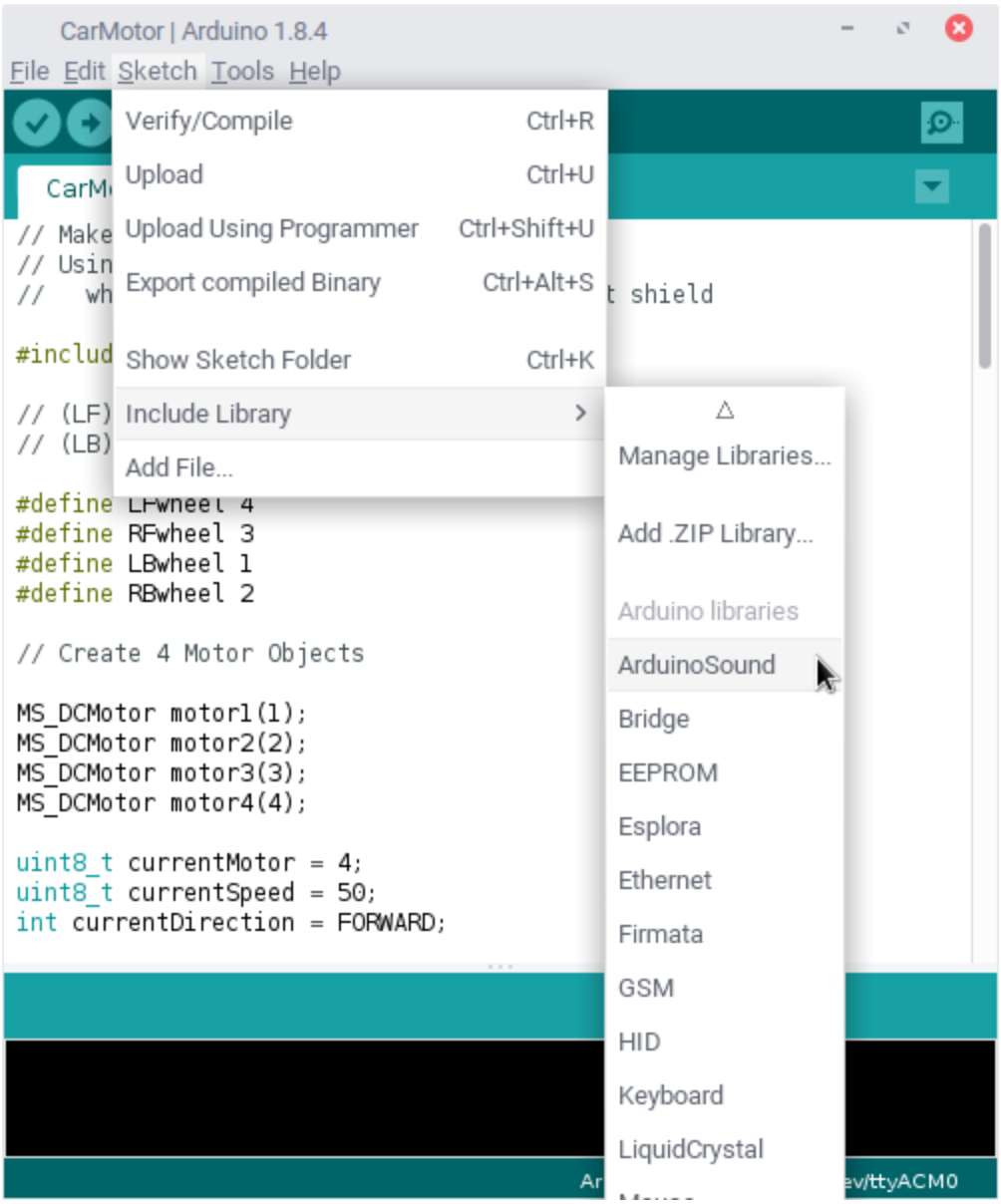


Once successfully installed the library manager will show the library as “Installed”



Installed libraries show up in the Include Library drop down menu.

Inline Task: Look at the Library menu before and after installing a library.
Look at the example menu.



Download ZIP Libraries

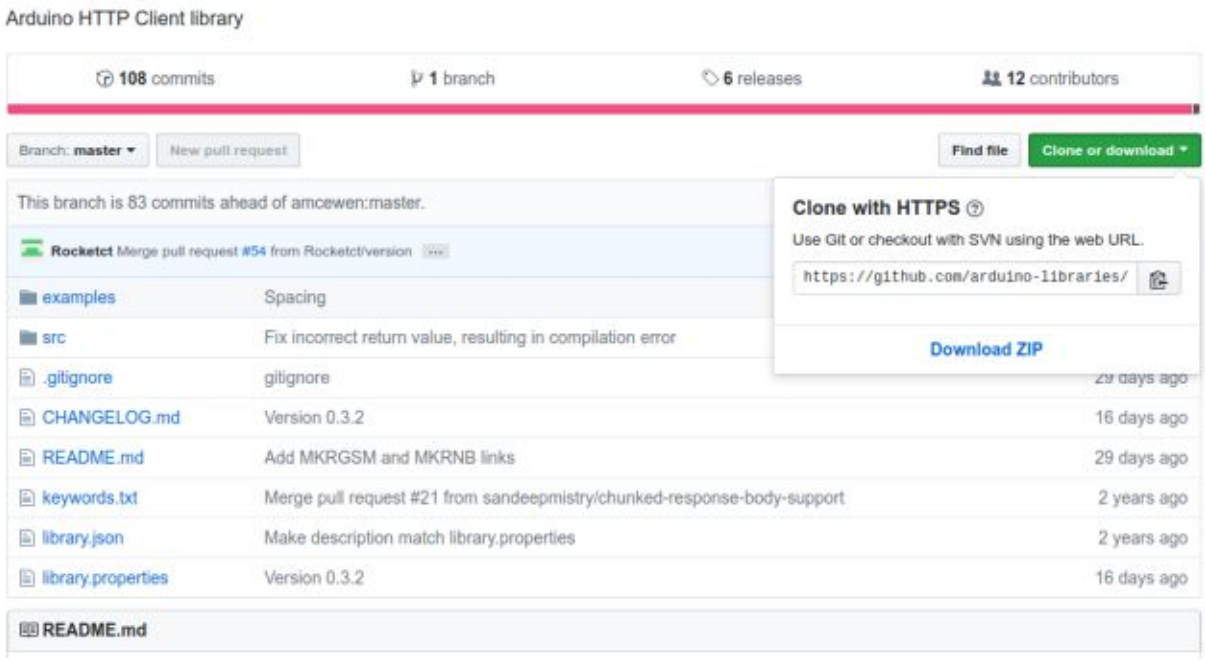
Functionality to support new electronic modules is usually downloaded from the Internet or created by the programmer.

A guide for installing libraries is: https://www.arduino.cc/en/Guide/Libraries

A common repository for Arduino libraries is GitHub.

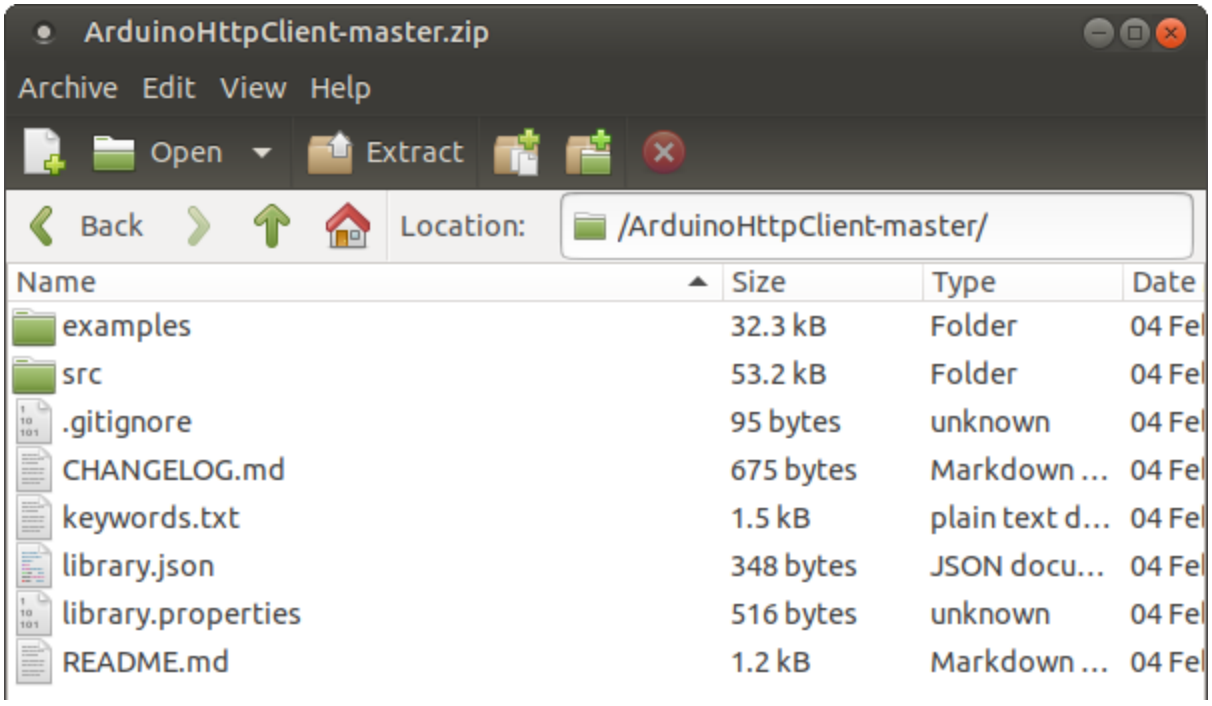
Here is an example library in GitHub

https://github.com/arduino-libraries/ArduinoHttpClient



Scrolling down on the page displays the documentation. Clicking on “Download ZIP” brings down the library ZIP file that can be imported into the Arduino IDE.

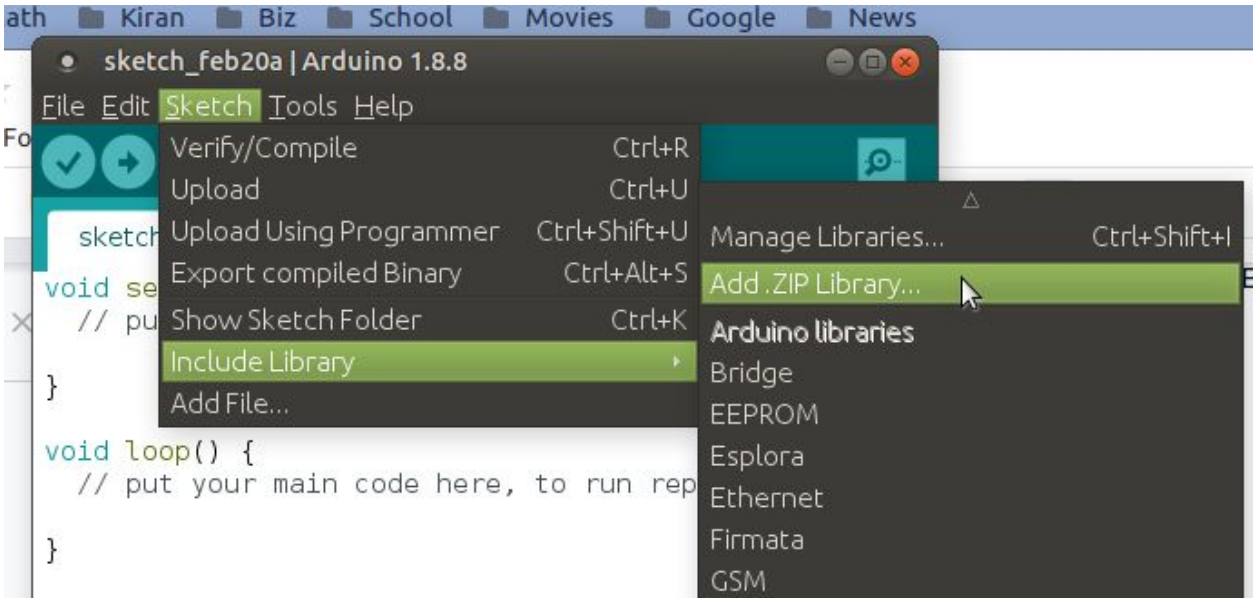
Here’s what’s inside the ZIP file. There is the source code, examples and documentation.



Notice that this particular library is not in the standard Arduino library format and may need to be re-organized before it can be imported.

Import Library into Arduino IDE

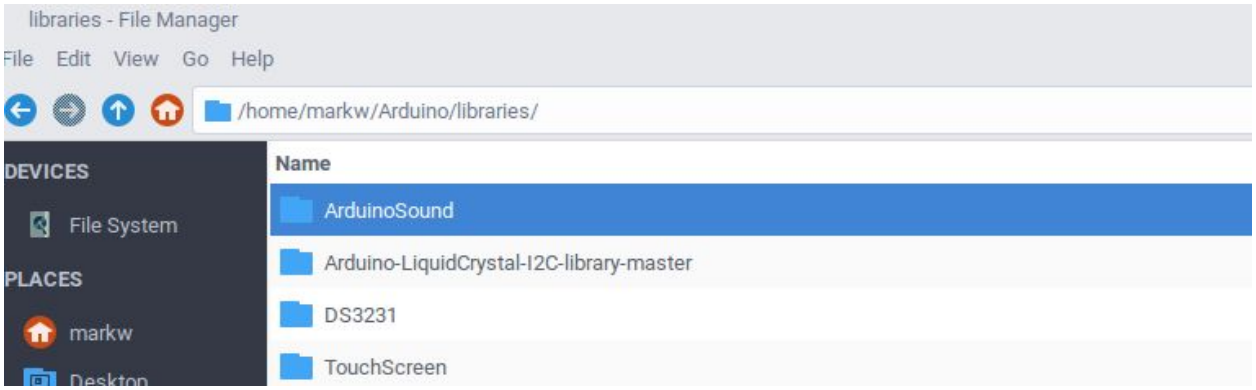
Once the ZIP file of a library is downloaded, it can be imported into the Arduino IDE using the Add .ZIP Library on the Sketch menu.



Once a library has been added, it will show up in the “Include Library” list under the Sketch menu.

Manual Installation

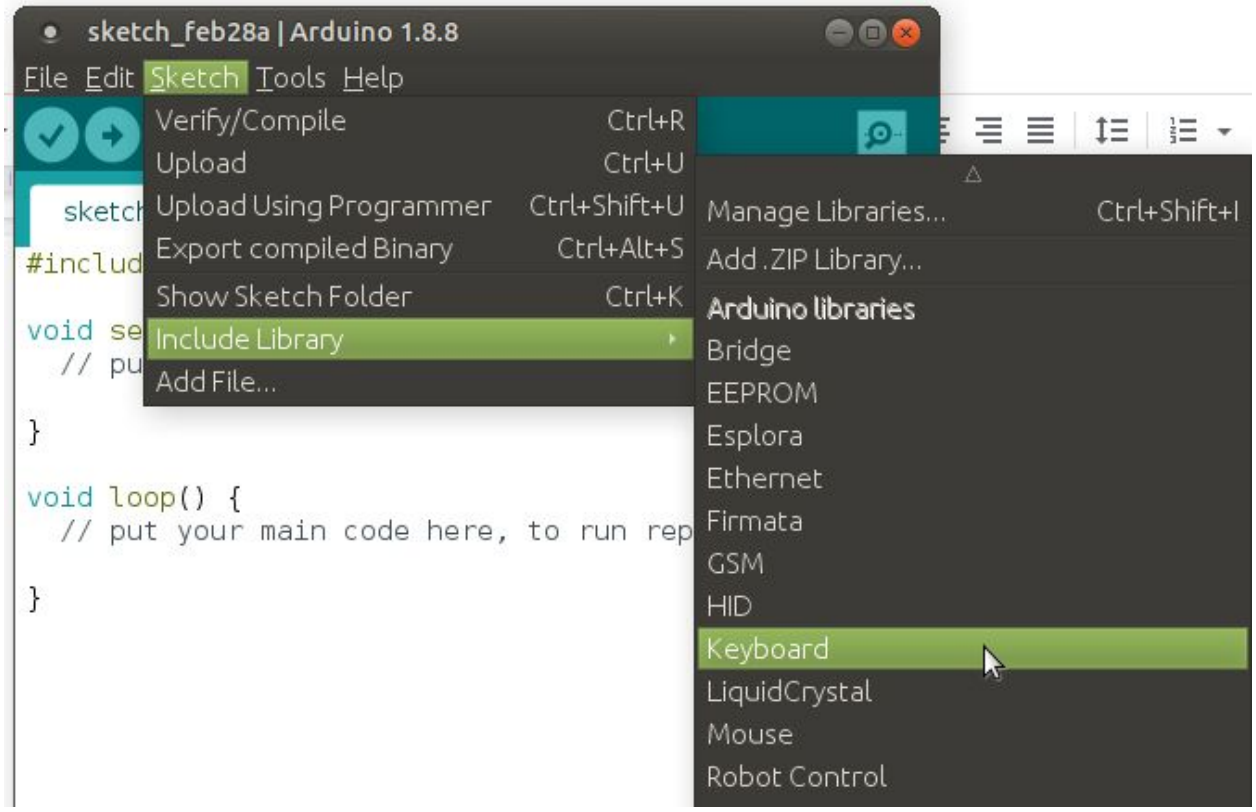
Libraries can be installed manually by unzipping the library file, then moving the files into the “libraries” folder in the Arduino folder where all the sketches have been stored.



When creating a custom library this is also the location to put the custom c++ and header files.

Compiling With a Library

Using a library in a *.INO sketch is simple. Just include the main *.h file. To find the name of the include file, use the include library menu and it will insert the correct include file for you. For example,



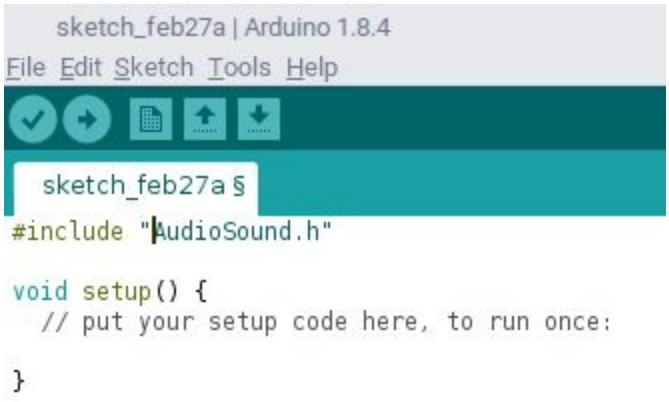
Will insert the line #include <Keyboard.h> at the top of the sketch. The header file can also be added manually if the programmer knows the name of the header to use.

When the sketch is compiled, the IDE searches for the associated *.h and *.cpp files and adds them into the compilation process.

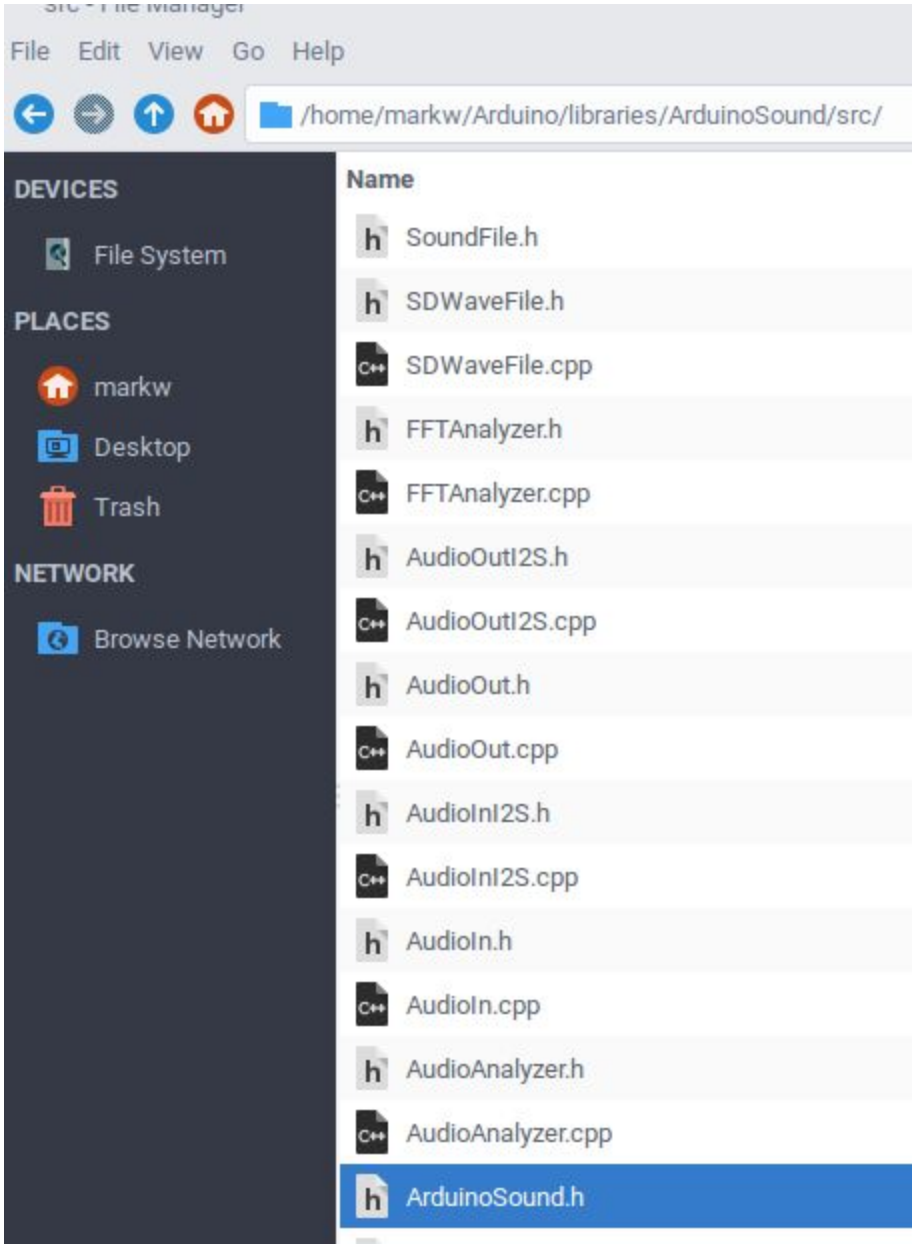
For example, to use the AudioSound library insert the line #include <AudioSound.h> at the top of the sketch manually or by clicking on the library name in the Include Library menu. Technically, libraries that have been installed in the Arduino/libraries/ folder should use the quote syntax
#include “AudioSound.h”

While libraries that are in the Arduino IDE directory should use #include <AudioSound.h> , however most compilers will look in both the IDE and local libraries directory.

In this example, the AudioSound library is in the Arduino/libraries directory



The file is in the libraries directory along with all the C++ source files.



C++ Basics

The C++ language extends the C language by adding objects. It groups functions and data into classes. This object oriented approach is quite valuable when there are specific objects in the system design. User interfaces are natural for the object oriented approach since there are objects like windows, keyboards, buttons, which must be created.

The Arduino IDE uses objects. For example, the Serial object has methods like “begin()” or “read()” and are called by Serial.begin(baud_rate).

The C++ language lets the programmer define classes. An instance of a class is an object. An object has private variables and functions (called methods) which are only visible to the object itself, and public methods and variables which are visible to other parts of the program.

```
// A class definition
class MySerialPort {
// private variables
    int baud_rate, TXpin, RXpin;

// public variables and methods
  public:
    void set_pins (int,int);
    Int getRX(void);
};

// Example of a method of the class MySerialPort
void MySerialPort:: set_pins(int rx, int tx) {
          RXpin = rx;
          TXpin = tx;
}
```

In the main program an instance of the MySerialPort is declared:
MySerialPort myPort;

```
// A method of the newly created object is called, for example, by:
myPort.set_pins( 9,10);
```

Inline Question: Name a standard Arduino library and a method of that library

The #include Servo.h includes the Servo class. A method of that class is myServo.attach(thePin)

Making a Library

The process of creating a library for the Arduino IDE is described at:
https://www.arduino.cc/en/Hacking/LibraryTutorial

A custom library entails writing a header file, a c++ program, and an example.

The header file describes the class. The cpp program file contains the methods. And an example file is good form to help the user.

---------- header File ----------
The header file must include “Arduino.h” and must not include itself multiple times. It should have a brief description at the start. The Serial_helper.h file starts below.

```
/*
*** Header files contain comments which explain the usage and purpose of the class ***
Serial_helper.h
A few utility programs to help convert characters to integers when read from the Serial.read()
*/
#ifndef Serial_helper_h
#define Serial_helper_h

#include "Arduino.h"

class Serial_helper
{
  public:
    Serial_helper();
          // converts a character to 0-9 or returns -1 if fail
    int toNumber();
          // tests if a character is a digit 0-9 or -1 if fail
    int isNumber();

};

#endif
```

(Note: the *.h file ends here)

---------- C++ File ----------
The Serial_helper.cpp file starts here. It is a separate file.

```
/*
  Serial_helper.cpp - Library for converting a character to a number
  Created by Mark Webster 2019
  Released into the public domain.
*/
```

```
#include "Arduino.h"
#include "Serial_helper.h"
#include <ctype.h>

// Returns 0-9 or -1 if it fails to convert.
int Serial_helper::toNumber(char oneChar)
{
	int theInteger = -1;
	if( oneChar >= '0' && oneChar <= '9' )
		theInteger = oneChar - '0';

	return theInteger;
}

// Returns -1 if not an integer
int Serial_helper::isNumber(char oneChar)
{
	int theValue = -1;

	theValue = oneChar - '0';

	if( theValue <0 || theValue > 9 ) theValue = -1;

	return theValue;

}
```

(End of Serial_helper.cpp file)

---------- Example File ----------
(Start of an example *.ino file. It is a separate file either in the examples folder or in a sketch folder in the Arduino folder)

A file that is put into an examples subfolder in the library folder

```
#include <Serial_helper.h>

Serial_helper sh();

void setup()
{
	Serial.begin(9600);
}
```

```
void loop()
{
	char oneChar;
	int theValue;

	if (Serial.available()) {
		oneChar = Serial.read();
		if( sh.isNumber( oneChar) ) {
			theValue = sh.toNumber( oneChar );
			Serial.println(theValue);
		}
	}
	delay(500);

}
```

(End of example Serial_helper.ino file.)

Exercises

Exercise 1: Type in and create the Serial_helper library.

Use an external editor like Pluma or Notepad++ to enter and edit the files. There will be 3 files created, a header, a cpp file, and and example *.ino file.

Put the files in the Arduino -> libraries -> Serial_helper folder. There will be a Serial_helper.h file, a Serial_helper.cpp file, and an example *.ino file in an “examples” subfolder.
Be sure to save the files as ACII not Unicode, otherwise there will be invisible non-printing characters in the source code files.