Arduino IDE

Written by Mark Webster, Ousema Zayati, and ??

Summary

Learn to use the Arduino IDE to develop microcontroller projects.

Introduction

The software called Arduino Integrated Development Environment (Arduino IDE) is free software from the Arduino organization used to create programs that run on the Arduino family of microcontroller boards, and many other common microcontroller boards as well.

Download

The Arduino IDE can be obtained from: https://www.arduino.cc/en/main/software

Download the Arduino IDE



ARDUINO 1.8.8 The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other opensource software. This software can be used with any Arduino board. Refer to the Getting Started page for Installation instructions. Windows Installer, for Windows XP and up Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10

Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits Linux 64 bits Linux ARM

Release Notes Source Code Checksums (sha512)

The IDE is cross platform and available on the three main operating systems, Windows, Mac OS X, and Linux. Most computers are 64 bit. Raspberry Pi computers are ARM based.

On Windows the IDE is installed at C:\Program Files (x86)\Arduino\ On Raspbian the IDE is usually installed at: /usr/share/arduino/ On other Linux systems the IDE may be in /opt/arduino/ or /usr/local/arduino/

Generally the full windows IDE is preferred over the Windows app.

The installer should put the files in the correct location. The Arduino sketches (programs) you create are stored in a different spot on the computer, as are the libraries you download. Generally, you don't need to know where the IDE stores its files. The menu or icon shortcut will have the path to the arduino executable.

The Arduino IDE is written in the Java language which interpreted so it is cross platform for any operating system with a Java Runtime Environment (JRE). That should be installed already with the operating system. Raspbian includes the hard float optimized version of the Java SE Runtime Environment.

If for some reason it isn't or is corrupt, then download a new version from Oracle. <u>https://www.java.com/en/download/</u>

On Linux, the version of the Arduino IDE that is installed by "sudo apt-get install arduino" is usually old. The newest version can be downloaded from the Arduino.cc website. For the Raspberry Pi, download the ARM version. Multiple versions of the Arduino IDE can be installed on the same computer if needed to work with older sketches.

Sketch and Library Location

The programs created with the Arduino IDE are called sketches, but they are actually C or C++ programs. The file extension is *.ino. The sketch is stored in a directory/folder with the same name as the *.ino file. The arduino sketches and corresponding folders are located in an Arduino folder for newer versions of the IDE.

On the Raspberry Pi for older versions of the Arduino IDE, like 1.6.6, these sketches are in a folder /home/pi/sketchbook. For example, the sketch KeyEcho.ino is located in /home/pi/sketchbook/KeyEcho/KeyEcho.ino. All the user installed libraries on the Raspberry Pi for older versions of the IDE go in the folder /home/pi/sketchbook/libraries.

On Linux, for newer versions of the Arduino IDE, like 1.8.8, the sketches and libraries are installed in a folder called "Arduino" in the user's home directory. For example, for the user "pi" the folder is /home/pi/Arduino/KeyEcho and the libraries are in a folder called "libraries" in the Arduino folder.

On Windows machines the sketches and libraries are in a folder called "Arduino" in the Documents folder. In Windows the Documents folder is in the "My Documents" folder, or for Windows 10 the location is ThisPC -> Documents -> Arduino

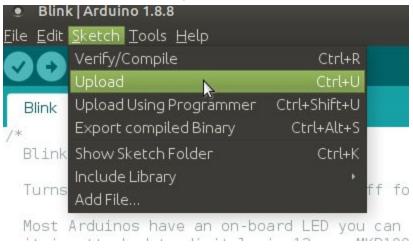
A guide to installing libraries is at: <u>https://www.arduino.cc/en/Guide/Libraries</u>

Compile and Upload Code

In the Arduino IDE there are two icons on the top row for compile and upload. The checkmark is compile, the right arrow is upload, the grid is new sketch, the up arrow is open, the down arrow is save.



The same commands are available on the Sketch menu. Note the keyboard shortcuts of Ctrl+R, Ctrl+U which are quite handy.



By default, the Upload command also compiles the code.

Inline Question: List 3 ways to compile a program

In the black console window at the bottom, there are any compile errors, warnings, and also info on how much memory was used. Since the Arduino Uno only has 32 K of RAM, it is necessary to manage memory usage for larger programs. Just click in the console window and Ctrl+A to select all the comments, then Ctrl+C to copy them. Paste into any text editor to view the messages. Here's an example of messages for the Blink program. There were no errors.

}	* 1
expected ';' before '}' token	Copy error messages
<pre>halogReadSerial:17:1: error: exp } </pre>	<pre>bected ';' before '}' </pre>

This is the full text pasted into the Pluma text editor. Notice the actual error is a missing semicolon several lines above the last error found by the compiler. It helps to scroll up in the error messages to the first error listed.

*Unsaved Document 1 - Pluma				•••
File Edit View Search Tools Documents Help				
📑 Open 👻 💆 Save 💾 🦐 Undo 🦽	X 🖻 💼	0 🕅		
🕒 *Unsaved Document 1 🗱				
1 Arduino: 1.8.8 (Linux), Board: "Arduino 2			tion lucid cotu	
<pre>3 /tmp/arduino_modified_sketch_858067/Ana 4 AnalogReadSerial:17:1: error: expected</pre>			tion void setu	p():
5 }				
<pre>6 ^ 7 exit status 1 8 expected ';' before '}' token 9 10 This second to world have note information</pre>				
10 This report would have more information with 11 "Show verbose output during compilation" 12 option enabled in File -> Preferences.				
13		3		
	Plain Text 🔻	Tab Width: 4 🔻	Ln 5, Col 3	INS

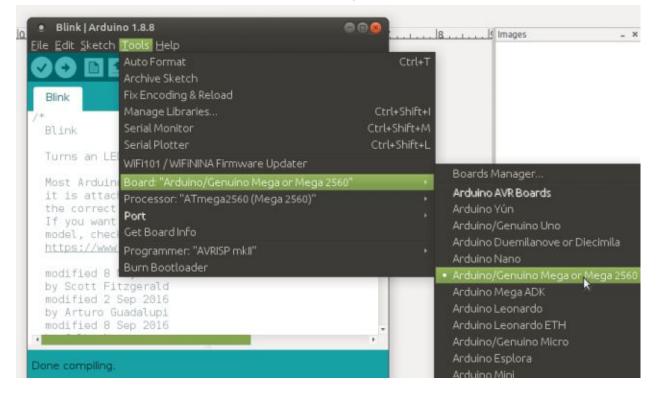
When successfully compiled, the black console window will describe the amount of memory used by each part of the code. The Arduino Uno only has 32K of flash ram so for larger programs this memory usage is critical information.

This BLINK sketch is about as tiny as it gets, and uses 1460 bytes (0%) of program storage space. Maximum is 253952 bytes.

Global variables use 9 bytes (0%) of dynamic memory, leaving 8183 bytes for local variables. Maximum is 8192 bytes.

Selecting Board and Serial Port

Prior to compiling a new sketch for the first time, one must select the Arduino board to be used and the serial port. The standard boards are listed. Clones use the same settings as the standard board. If additional boards are installed they will show up in the sub-menu.



Select the port the Arduino cable is connect to. On Windows serial ports are called "COM1:, COM2:, etc". On Linux serial ports are files in the "/dev/" directory. If you are unsure which port is used, in a terminal window before plugging in the Arduino to the USB type "Is /dev/tty*" then plug in the Arduino and type "Is /dev/tty*" again. The new entry will be the Arduino. Here's and example of the terminal screen.

File Edit	Tabs Help				
pi@makerpi:	~ \$ ls /dev/	tty*			
/dev/tty	/dev/tty19	/dev/tty3	/dev/tty40	/dev/tty51	/dev/tty62
/dev/tty0	/dev/tty2	/dev/tty30	/dev/tty41	/dev/tty52	/dev/tty63
/dev/tty1	/dev/tty20	/dev/tty31	/dev/tty42	/dev/tty53	/dev/tty7
/dev/tty10	/dev/tty21	/dev/tty32	/dev/tty43	/dev/tty54	/dev/tty8
/dev/tty11	/dev/tty22	/dev/tty33	/dev/tty44	/dev/tty55	/dev/tty9
/dev/tty12	/dev/tty23	/dev/tty34	/dev/tty45	/dev/tty56	/dev/ttyAMA0
/dev/tty13	/dev/tty24	/dev/tty35	/dev/tty46	/dev/tty57	/dev/ttyprintk
/dev/tty14	/dev/tty25	/dev/tty36	/dev/tty47	/dev/tty58	
/dev/tty15	/dev/tty26	/dev/tty37	/dev/tty48	/dev/tty59	
/dev/tty16	/dev/tty27	/dev/tty38	/dev/tty49	/dev/tty6	
/dev/tty17	/dev/tty28	/dev/tty39	/dev/tty5	/dev/tty60	
/dev/tty18	/dev/tty29	/dev/tty4	/dev/tty50	/dev/tty61	
pi@makerpi:					
/dev/tty	/dev/tty19	/dev/tty3	/dev/tty40	/dev/tty51	/dev/tty62
/dev/tty0	/dev/tty2	/dev/tty30	/dev/tty41	/dev/tty52	/dev/tty63
/dev/tty1	/dev/tty20	/dev/tty31	/dev/tty42	/dev/tty53	/dev/tty7
/dev/tty10	/dev/tty21	/dev/tty32	/dev/tty43	/dev/tty54	/dev/tty8
/dev/tty11	/dev/tty22	/dev/tty33	/dev/tty44	/dev/tty55	/dev/tty9
/dev/tty12	/dev/tty23	/dev/tty34	/dev/tty45	/dev/tty56	/dev/ttyACM0
/dev/tty13	/dev/tty24	/dev/tty35	/dev/tty46	/dev/tty57	/dev/ttyAMA0
/dev/tty14	/dev/tty25	/dev/tty36	/dev/tty47	/dev/tty58	/dev/ttyprintk
/dev/tty15	/dev/tty26	/dev/tty37	/dev/tty48	/dev/tty59	
/dev/tty16	/dev/tty27	/dev/tty38	/dev/tty49	/dev/tty6	
/dev/tty17	/dev/tty28	/dev/tty39	/dev/tty5	/dev/tty60	
/dev/tty18	/dev/tty29	/dev/tty4	/dev/tty50	/dev/tty61	
	~ S scrot -u			,	

Select the serial port on the Tools -> Port menu.

Ctrl+T trl+Shift+I I+Shift+M rl+Shift+L	î	
trl+Shift+I I+Shift+M	î	
l+Shift+M	4	
l+Shift+M		
l+Shift+M	d	
	đ	
rl+Shift+L	2	
>	- -	
>	S	enal ports
	/0	dev/ttyACM0 (Arduino/Genuino Uno
>	/0	dev/ttyAMA0
		and the second second
	>	Contraction of the second
	>	/0

The Mac OS X operating system is a form of Unix, so the serial port can be found the same way as a Raspberry Pi. In a terminal window just type "Is /dev/tty*" before and after plugging in the cable.

Notice the Programmer item on the Tools menu is set to AVRISP mkil. This is the default unless using the Arduino as an ISP to program other microcontrollers.

Compile

The compilation process is simple for the user, as it was designed to be. Under the hood the IDE does some complicated actions. The complete build process can be found in: https://github.com/arduino/Arduino/wiki/Build-Process

Basically, the *.ino filenames are made into *.cpp, the IDE searches for files in the Arduino application folder, and in the user's library folder. It also compiles any system files needed in the IDE installation folder. The cross-compilers avr-gcc and avr-g++ invoked. Compiled object *.o files are linked together into a *.hex file which is uploaded to the Arduino board by the software "avrdude".

Some non-Arduino boards like ESP32 use slow Python code to compile Arduino sketches so the compile and upload process will take much longer than Arduino boards and their clones.

The larger the program, and the larger the included libraries, the longer the compile time.

Inline question: Name 3 things the IDE does when a sketch is compiled.

Upload

The upload button or keyboard shortcut Ctrl+U will compile and upload the sketch to the Arduino board on the specified serial port. If successful the message "Done uploading" will appear. The Arduino board serial TX and RX lights will blink. When done uploading the board will reset and the program will start running. Occasionally the reset fails, especially on other boards like the ESP32, and one must manually press the physical reset button on the board.

If the serial port is messed up somehow, it generate an error message. Sometimes rebooting the computer will reset the serial port. Sometimes the error is real and the board is defective or the cable is defective.

Under the hood, the program "avrdude" is called by the IDE to upload the compiled sketch.

Brave or crazy programmers can call avr-gcc and avr-dude manually within a terminal window and not use the Arduino IDE. There is a tutorial on avrdude at the ladyada website (related to Adafruit devices).

http://www.ladyada.net/learn/avr/avrdude.html

Using Libraries

The Arduino IDE is designed to be as friendly as possible to hobbyist makers. Often this means hiding the messy details of controlling and communication with sensors and electronic modules. The messy details are put in a library that the hobbyist can call without understanding the internals.

A few libraries are built-into the Arduino IDE, most are downloaded from the Internet and installed with the Library Manager.

Built in libraries

The built in libraries can be called without downloading and installing anything from the Internet. Details can be found at:

https://www.arduino.cc/en/Reference/Libraries

----Here's the list of default libraries as of Arduino IDE 1.8.8 -----

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.

Example code

Most libraries, even those downloaded from the Internet will contain example programs that are added to the Arduino IDE File-> examples menu toward the bottom. Custom library examples are at the far bottom of the menu. These are valuable to figure out how to use a library.

AnalogReadSerial Ardu Eile Edit Sketch Tools Hel		9 9
New Ctrl+N Open Ctrl+C Open Recent Sketchbook	09.USB	
<pre>Se Examples te Close Ctrl+W fo Save Ctrl+S do Save As Ctrl+Shift+S Page Setup Ctrl+Shift+F Preferences Ctrl+Comma Quit Ctrl+Q Preferences Ctrl+Comma Quit Ctrl+Q fo // the setup routine r void setup() { // initialize serial Serial.begin(9600); } ti Done compiling. Sketch uses 1868 bytes</pre>	Bridge Esplora Ethernet Firmata GSM LiquidCrystal Robot Control Robot Motor SD Servo SpacebrewYun Stepper Temboo RETIRED	uino
Global variables use 1 Global variables use 1 ated. A common way to solve this e and see the correct syntax for t	EEPROM SoftwareSerial SPI Wire Examples from Custom Libraries Bounce2 MCUFRIEND_kbv RTClib	LCD_ID_readreg readpixel_kbv scroll_kbv showBINP_kbv_as7 testcard_kbv Touch_shield_kbv UTouchScreen_Calibr_kbv UTouch_Calibr_kbv prary xample

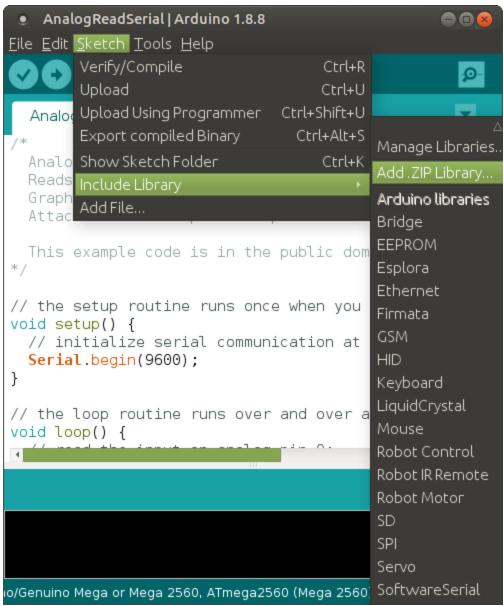
Incompatibilities

A common problem with examples or libraries downloaded from the Internet is incompatibility between the library and the example code. The library may have changed since the examples were created. A common way to solve this problem is to look at the source code for the library include file and see the correct syntax for the method calls for that library. Then edit the example code to match the actual library code.

Library Manager

To install new libraries, they can be copied manually into the libraries folder or preferably, installed using the Library Manager which is on the Tools menu.

To include an already installed library, use the Include Library menu item. This is not needed for standard libraries like <Wire.h>, <Servo.h>, <Stepper.h> which are installed by default.



Including the library puts the correct #define <libraryname.h> file at the start of the *.INO sketch. The header file can also be included manually.

If the library is not already installed, there are two choices. First option, go to the Manage Libraries and search for a new library using a keyword.

In the Library Manager dialog, one can search through the existing libraries for a term such as DH11. If the library is not already installed, click on the install button.

• Library Man	ager	den telebre						and the	
Type All	v	Topic	All	*	DH11				
	ead the ten	peratur				nsors. The library DH11, DHT21, DH	allows to obtain data o π22.	f relative humidity and	Install

Once installed, the text "INSTALLED" will appear whenever searching for that library. Example code to use that library will also be installed.

The second method is to download a library as a zip file, often from a location such as GitHub.

Here is an example of a DH11 library in GitHub. Click the "Download ZIP" button.

This code by	Arduino uno	board to run	DH11 sensor	with LCD1602_I2C
--------------	-------------	--------------	-------------	------------------

② 2 commits	I branch	© 0 releases	ases 1 contributor	
Branch: master - New pull request			Find file Clone or down	loa
eagi1 Add files via upload		Clone with	h HTTPS 💿	
🤤 .gitignore	Initial commit	Use Git or ch	eckout with SVN using the web URL.	
LCD1602_I2C.cpp	Add files via uploa	https://g	ithub.com/eagl1/DH11_LCD1602	隐
DLCD1602_12C.h	Add files via uploa	id a	Provide the Real Provid	
main program.ino	Add files via uploa	d	Download ZIP	

Once the ZIP file is downloaded, go to the Sketch->Include Library->Add .ZIP library option. Navigate to where the ZIP file is located, select it and click OK

Select a zip file or a folder c	ontaining the library you'd like to add 🛛 🔗				
New Folder Delete File Ren	ame File nome/markw/Downloads 💌				
Folders ./ / 3.5inch_Arduino_ILI9486_V1.0 BluesBox/ Clip-on_Marker_Holder/ Dons/	Eiles 2018-10-09-raspbian-stretch.zip Archive-8f87.zip Arduino-master.zip Clip-on_Marker_Holder.zip DH11_LCD1602_I2C-master.zip OscopeTouchMega.zip URTouch.zip UTFT.zip Writescad.zip				
<u>S</u> election: /home/markw/Dow	nloads				
DH11_LCD1602_I2C-master.zip					
Filter: ZIP files or folders					
	😣 Cancel 🛛 🖌 🧏				

Once the ZIP file is added, then the library can be included from the Sketch->Include Library menu.

AnalogReadSerial Arduino 1.8.8 Eile Edit Sketch Tools Help	
Verify/Compile Ctrl+R Upload Ctrl+U	P
AnalogUpload Using ProgrammerCtrl+Shift+U/*Export compiled BinaryCtrl+Alt+SAnaloShow Sketch FolderCtrl+KReadsInclude Library•Attac	▲ Ethernet Firmata GSM HID Keyboard
This example code is in the public dom */	LiquidCrystal Mouse Robot Control
<pre>// the setup routine runs once when you void setup() { // initialize serial communication at Serial.begin(9600); }</pre>	Robot Control Robot IR Remote Robot Motor SD SPI
<pre>// the loop routine runs over and over a void loop() {</pre>	Servo SoftwareSerial SpacebrewYun Stepper TFT
	Temboo WiFi Wire
io/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560)	Contributed libraries Adafruit_TFTLCD Bounce2 DH11_LCD1602_I2C-ngaster MCUFRIEND_kbv RTClib Sodag_DS3231

Once a library is included, then it can be referenced inside a sketch using the name #include <DH11_LCD1602_I2C-master.h>

The actual location of the library is in the libraries folder inside the user's Arduino folder.

To write your own libraries, review the documentation at: <u>https://www.arduino.cc/en/Hacking/libraryTutorial</u>

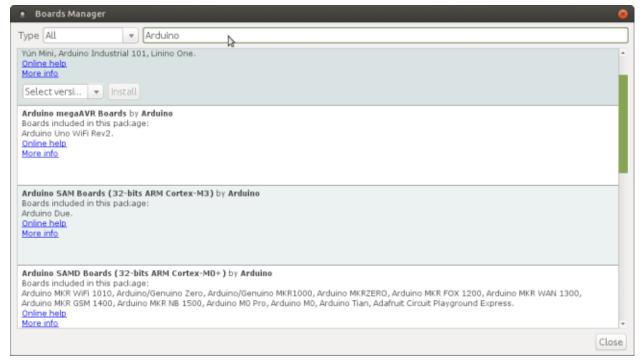
There is a style guide for writing https://www.arduino.cc/en/Reference/APIStyleGuide

Inline question: How to access a library in a sketch?

Board Manager

Additional non-Arduino boards (called cores) can be programmed using the Arduino IDE. Details are given at: <u>https://www.arduino.cc/en/quide/cores</u>

If a board has been imported already, it can be found in the board manager by searching for that board name. Just click the Install option.



If the board has not been imported, use the Preferences menu Additional board URL. The JSON url is found from the board manufacturer.

Preferences		×
Settings Network		
Sketchbook location:		
C:\Users\Simone\Documents	\Arduino	Browse
Editor language:	English (English)	(requires restart of Arduino)
Editor font size:	16	
Interface scale:	Automatic 100 -% (requires restart of Arduino)
Show verbose output during:	compilation upload	
Compiler warnings:	None 🗸	
Display line numbers		
Enable Code Folding		
Verify code after upload		
Use external editor		
Check for updates on star	tup	
Update sketch files to nev	v extension on save (.pde -> .ino)	
Save when verifying or up	bloading	
Additional Boards Manager UF	RLs: package_esp8266com_index.json,http://digistump.co	m/package_digistump_index.json
More preferences can be edit	ed directly in the file	
C:\Users\Simone\AppData\Lo	cal\Arduino15\preferences.txt	
(edit only when Arduino is not	running)	
		OK Cancel

For example, to add the ESP32 board use the URL: <u>https://dl.espressif.com/dl/package_esp32_index.json</u>

To add the ATTiny boards use the URL:

https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_ attiny_index.json

To enter multiple JSON URLs click on the icon next to the line with the Additional Board Manager URLs. A dialog box pops up with can support multiple lines of URLs.

Preferences		77	0
Settings Network			
Sketchbook location:			
/home/markw/Arduino			Browse
Editor language:	System Default	•	(requires restart of Arduino)
Editor font size:	12		
Interface scale:	🗹 Automatic 🔢 🕄 🛠 (requires restart of Arc	duino)	
Theme:	Default theme 💌 (requires restart of Arduino)		
Show verbose output during:	 Additional Boards Manager URLs 		8
Compiler warnings:	Enter additional URLs, one for each row		
Display line numbers	https://raw.githubusercontent.com/damellis/attin https://dl.espressif.com/dl/package_esp32_index.j		.6.x-boards-mana
Enable Code Folding Verify code after upload			
🗍 Use external editor	•		
🛛 Aggressively cache compil	Click for a list of unofficial boards support URLs		
🗹 Check for updates on star			OK Cancel
🛛 🗹 Update sketch files to nev	execusion on advertapoer in anoy		
Save when verifying or uplo	ading		
Additional Boards Manager URL	s: [s/attiny/ide-1.6.x-boards-manager/package_da	mellis_	attiny_index.json
More preferences can be edite	d directly in the file		13

The board can then be installed with the Board Manager.

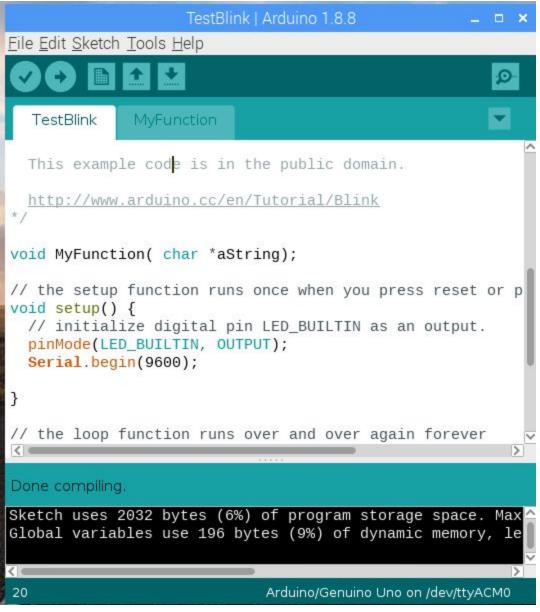
🜻 Boards Mana	ger	8
Type All	▼ attiny	
Boards included in	n Mellis version 1.0.2 INSTALLED n this package: 5, ATtiny44, ATtiny84.	Remove

For example, here is the ATTiny board after it is installed.



Additional C/C++ Code

An Arduino sketch can be broken up into several files. They must all end in *.ino and the main file must have the same name as the sketch directory. Each separate sketch ino file will appear as a different tab in the editor window of the IDE.



Functions in the second ino file should be declared in the primary ino sketch. The second ino file can reference default objects like Serial that were declared in the primary sketch.

Other C++ files and include files (*.cpp and *.h) generally are made into libraries and #include "mylibrary" is used to include the custom library.

Debugging

Compile time errors are debugged using messages in the black console window at the bottom of the IDE.

The most common method to debug an executing program is to put Serial.println(a variable) into the code and look at the values in the Serial Monitor. The debug code can be removed later. Sprinkling delay() statements in the code can indicate whether a process was reached or completed.

Code can often be debugged with an online Arduino Simulator. Just search for "online Arduino simulator" for links. An example is <u>https://circuits.io/</u>

Microsoft Visual Studio with extensions installed can debug Arduino code, assuming an Arduino is attached.

Serial Monitor

The serial monitor can be opened from the Tools menu of the IDE.

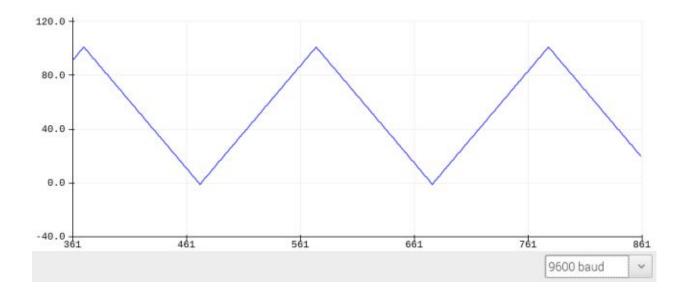
Norr		
Blink Arduin	no 1.8.8	000
<u>F</u> ile <u>E</u> dit <u>S</u> ketch	<u>T</u> ools <u>H</u> elp	
OO DE	Auto Format	Ctrl+T
	Archive Sketch	
Blink	Fix Encoding & Reload	
/*	Manage Libraries	Ctrl+Shift+I
Blink	Serial Monitor	Ctrl+Shift+M
	Serial Plotter	Ctrl+Shift+L
Turns an LEI	WiFi101 / WiFiNINA Firmware Updater	
Most Arduin	Board: "Arduino/Genuino Mega or Mega 2560	*
it is attacl	FIOCESSOL ATHICIDESOU (MCDB 2500)	•
the correct If you want	Doct	۰.
model, check		
https://www	Programmer: "AVRISP mkll"	
modified 8	Burn Bootloader	

The Serial Monitor window has a text entry box at the top where numbers or letters can be entered and sent to the Arduino board by pressing the Send button. The lower large display area is for text sent back from the Arduino board to the Arduino IDE.

Stuff to send to Arduino	Ser	nd
Blink 48		10
Blink 49		
Blink 50		
Blink 51		
Blink 52		
Blink 53		
Blink 54		
Blink 55		
Blink 56		
3link 57		
Blink 58		
Blink 59		
3link 60		
Blink 61		
Blink 62		
		Y
🛃 Autoscroll 🗌 Show timestamp	Newline v 9600 baud v Clear outp	ut

Serial Plotter

For numeric data sent back from the Arduino board, the values can be graphed with the serial plotter. Open the Serial Plotter from the Tools menu (just below the Serial Monitor). The Serial Plotter automatically rescales the vertical axis which can be both confusing and useful.



Arduino as ISP

For programming non-Arduino boards, a regular Arduino board can be used to program the non-Arduino board. Full details are at:

https://www.arduino.cc/en/tutorial/arduinoISP

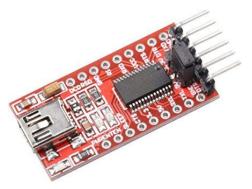
The menu item is Tools -> Programmer -> Arduino as ISP

Normally the programmer is set to AVRISP mkll, and it must be changed.

n	Movies a	loogle mews			
	Blink Arduin	no 1.8.8	008		122
	<u>Eile E</u> dit <u>S</u> ketch	Tools <u>H</u> elp		Q	SIGN
	AA DI	Auto Format	Ctrl+T		
		Archive Sketch			
	Blink	Fix Encoding & Reload		LP	
	/*	Manage Libraries	Ctrl+Shift+I		
	Blink	Serial Monitor	Ctrl+Shift+M		
	-	Serial Plotter	Ctrl+Shift+L		
C	Turns an LE	WiFi101 / WiFiNINA Firmware Updater			
-	Most Arduin	Board: "Arduino/Genuino Uno"			
	it is attac	Port			
ar	the correct If you want	Get Board Info		luino board is	
So		Programmer: "Arduino as ISP"		AVR ISP	
tra	https://www	Burn Bootloader		AVRISP mkll	
L.I.	modified 8	May 2014		USBtinyISP	
_	by Scott Fi	tzgerald		ArduinoISP	
	modified 2			ArduinolSP.org	
	by Arturo G modified 8			USBasp	
	(56p 2010		Parallel Programmer	
		119	_	 Arduino as ISP 	N
				Arduino Gemma	
1	5			BusPirate as ISP	
				Atmel STK 500 develop	oment board
				Atmel JTAGICE3 (ISP n	
		Arduino/Genuino Uno on /	dev/ttvACM1	Atmel JTAGICE3 (JTAC	Gmode)
1				Atmel-ICE (AVR)	100

Usually some extra electronics components must be connected to the Arduino to communicate with the other board.

Instead of using an Arduino as an IDE, one can use a FTDI USB breakout connected to the new chip. For example, ATTiny chips do not have a USB connection and must use an ISP or FDTI breakout. Some boards like the ESP32 have a USB connection and don't need ans ISP.



If a person is making their own Arduino board from an ATMEL microcontroller chip, they must burn the Arduino bootloader into the chip. This is also done from the Tools -> Programmer menu.

Preferences

There are some preferences that can be set from the File -> Preferences dialog.

Preferences	9			
Settings Network				
Sketchbook location:				
/home/markw/Arduino	Browse			
Editor language: 🛛 🍃	System Default v (requires restart of Arduino)			
Editor Font size:	12			
Interface scale:	Automatic 100 🗘 % (requires restart of Arduino)			
Theme:	Default theme 💌 (requires restart of Arduino)			
Show verbose output during:	Compilation Upload			
Compiler warnings:	None 💌			
Display line numbers				
Enable Code Folding				
Verify code after upload Use external editor				
Aggressively cache compile	d core			
🗹 Check for updates on start	qu			
🗹 Update sketch files to new	extension on save (.pde -> .ino)			
Save when verifying or uploading				
Additional Boards Manager URI	.5:			
More preferences can be edite				
/home/markw/.arduino15/pre				
(edit only when Arduino is not running)				
	OK Cancel			

For example, the location of all the Arduino sketches can be changed. The editor font size can be set. The number of compiler warnings can increased. And an external editor can be used with the Arduino IDE.

Adding new boards can be done on this dialog box. For example, for the Expressif ESP32 board, the URL

https://dl.espressif.com/dl/package_esp32_index.json

See the Board Manager section of this document for more details.

If an external editor is used, the IDE edit window will be gray and all edits are done in the external editor. The IDE will just be used to compile and upload the code.

Examples

Example 1: Change Arduino Uno to Nano

Hardware: Arduino Uno, Arduino Nano, USB-A, USB-mini cables *Software*: Arduino IDE

Find the serial port attached to an Arduino Uno. Write a simple program to blink an LED on pin 12, then run it on the Arduino Uno. You can also edit the BLINK example. Replace the Uno with a Nano (and change the cable from USB-A to USB-mini).

Upload the same program to the Nano. The Nano pins are standard spacing (unlike the Arduino Uno) so it can be put into a regular breakout board.

Note 1: on the Tools menu, change the board. Often the port must be changed as well. Note 2: Sometimes clone boards like the Nano will use the old bootloader. So if there is an error when uploading select the older processor on the Tools menu.

• Blink Ardu	ino 1.8.8	000		۰ ۱	
Eile Edit Sketcl	h Tools Help			x	0 - ~
	Auto Format Archive Sketch	Ctd+T	7 1		-
e Blink	Fix Encoding & Reload Manage Libraries	Ctrl+Shift+I			
Blink	Serial Monitor Serial Plotter	Ctrl+Shift+M Ctrl+Shift+L			
r Turns an L n	WiFi101 / WiFiNINA Firmware Updater				
	n: Board: "Arduino Nano" ^{CI} Processor: "ATmega328P (Old Bootloader	۲ ۲۵	[mega328	3P	_
the correc If you wan model, che	Port: "/dev/ttyUSB0"	• • A1		3P (Old E	Bootloader)
	Programmer: "Arduino as ISP"	+	rinegorot		
a modified 8	Burn Bootloader				

Example 2: Serial Monitor and Plotter

Hardware: breadboard, potentiometer, jumper cables *Software*: Arduino IDE

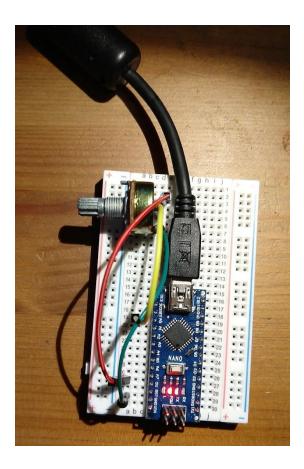
Connect a potentiometer to an Arduino Nano A0 analog input. Every 10 milliseconds read the value with analogRead(A0) and write it to the Serial output with Serial.println(). Be sure to initialize the Serial port with Serial.begin() in the loop() section. The AnalogReadSerial example can be edited if desired.

View the output in the Serial Monitor and in the Serial Plotter which are selected from the Tools menu.

Note 1: Sometimes the pins of the Nano must be spread wider to fit into the breadboard.

Note 2: older versions of the Arduino IDE do not have the Serial Plotter.

Note 3: be sure the baud rate is the same for the board, the serial monitor, and the plotter.



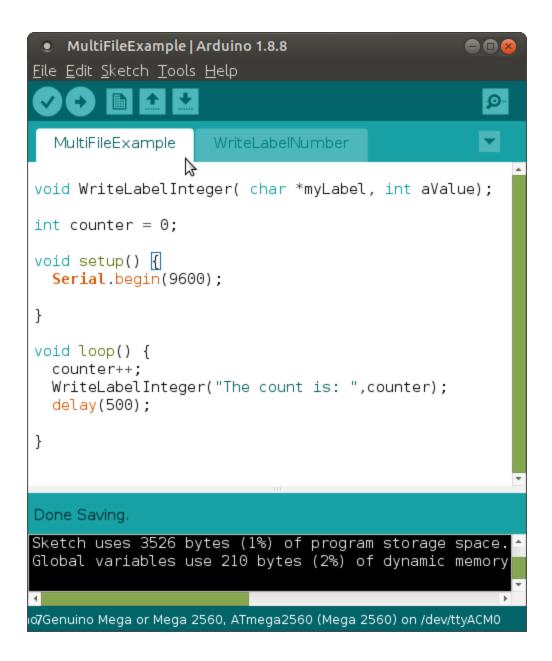
Example 3: Second INO File

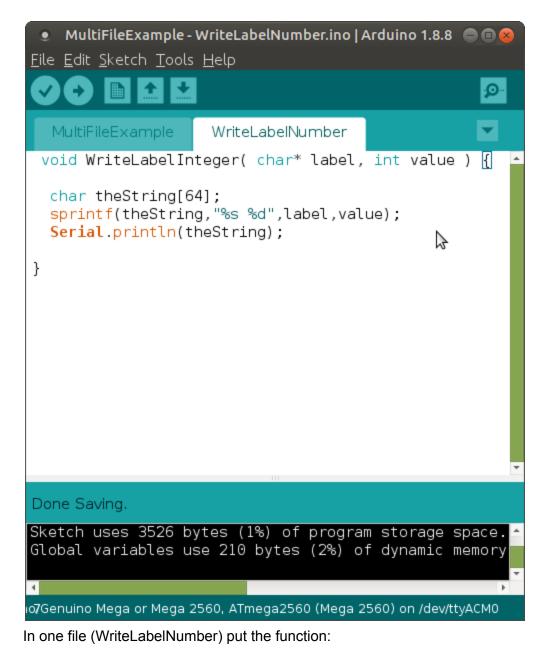
Hardware: Arduino Mega, USB-A cable *Software*: Arduino IDE

Use an Arduino Mega. No special hardware wiring is required. Create a function that writes a numeric value to the Serial output with a label. Put that function in a separate INO file in the same directory as your main Arduino sketch. Call that separate function from the main INO sketch.

Create a new INO file for the new function but save it into the same folder as the main INO sketch. All INO files must be in the same folder. Sometimes it is easier to initially create the second INO file in a text editor outside the IDE.

If the files are in the right location they will appear as separate tabs in the IDE editor.





void WriteLabelInteger(char* label, int value) {
 char theString[64];
 sprintf(theString,"%s %d",label,value);
 Serial.println(theString);

}

In the main file (MultiFileExample) declare the function:

```
void WriteLabelInteger( char *label, int value);
int counter = 0;
```

```
setup() {
    Serial.begin(9600);
}
loop() {
    counter++;
    WriteLabelInteger( "The count is: ", counter);
    delay(500);
}
```

View the output with the serial monitor.