**Arduino Development Environment**

The Arduino development environment contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions, and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

**Writing Sketches**

Software written using Arduino are called **sketches**. These sketches are written in the text editor. Sketches are saved with the file extension .ino. It has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino environment including complete error messages and other information. The bottom righthand corner of the window displays the current board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

**NB: Versions of the IDE prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the .ino extension on save.**

|  |  |
| --- | --- |
|  | *Verify*  Checks your code for errors. |
|  | *Upload*  Compiles your code and uploads it to the Arduino I/O board. See [uploading](http://arduino.cc/en/Guide/Environment?from=Tutorial.Bootloader#uploading) below for details.  Note: If you are using an external programmer, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer" |
|  | *New*  Creates a new sketch. |
|  | *Open*  Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window.  Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the **File | Sketchbook** menu instead. |
|  | *Save*  Saves your sketch. |
|  | *Serial Monitor*  Opens the [serial monitor](http://arduino.cc/en/Guide/Environment?from=Tutorial.Bootloader#serialmonitor). |

Additional commands are found within the five menus: **File**, **Edit**, **Sketch**, **Tools**, **Help**. The menus are context sensitive which means only those items relevant to the work currently being carried out are available.

**Edit**

* *Copy for Forum*   
  Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
* *Copy as HTML*   
  Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.

**Sketch**

* *Verify/Compile*   
  Checks your sketch for errors.
* *Show Sketch Folder*   
  Opens the current sketch folder.
* *Add File...*   
  Adds a source file to the sketch (it will be copied from its current location). The new file appears in a new tab in the sketch window. Files can be removed from the sketch using the tab menu.
* *Import Library*   
  Adds a library to your sketch by inserting #include statements at the code of your code. For more details, see [libraries](http://arduino.cc/en/Guide/Environment?from=Tutorial.Bootloader#libraries) below. Additionally, with version 1.0.5 and later of the IDE, you can import a library from a .zip file.

**Tools**

* *Auto Format*   
  This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
* *Archive Sketch*   
  Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
* *Board*   
  Select the board that you're using. See below for [descriptions of the various boards](http://arduino.cc/en/Guide/Environment?from=Tutorial.Bootloader#boards).
* *Serial Port*   
  This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.
* *Programmer*   
  For selecting a harware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're [burning a bootloader](http://arduino.cc/en/Tutorial/Bootloader) to a new microcontroller, you will use this.
* *Burn Bootloader*   
  The items in this menu allow you to burn a [bootloader](http://arduino.cc/en/Tutorial/Bootloader) onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino board but is useful if you purchase a new ATmega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the **Boards** menu before burning the bootloader.

**Sketchbook**

The Arduino environment uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the **File > Sketchbook** menu or from the **Open** button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the **Preferences** dialog.

'''Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino.

**Tabs, Multiple Files, and Compilation**

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no extension), C files (.c extension), C++ files (.cpp), or header files (.h).

**Uploading**

Before uploading your sketch, you need to select the correct items from the **Tools > Board** and **Tools > Serial Port** menus. The [boards](http://arduino.cc/en/Guide/Environment?from=Tutorial.Bootloader#boards) are described below. On the Mac, the serial port is probably something like **/dev/tty.usbmodem241** (for an Uno or Mega2560 or Leonardo) or **/dev/tty.usbserial-1B1** (for a Duemilanove or earlier USB board), or **/dev/tty.USA19QW1b1P1.1** (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably **COM1** or **COM2** (for a serial board) or **COM4**, **COM5**, **COM7**, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be **/dev/ttyUSB0**, **/dev/ttyUSB1** or similar.

Once you've selected the correct serial port and board, press the upload button in the toolbar or select the **Upload** item from the **File** menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino environment will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino **bootloader**, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

**Libraries**

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the **Sketch > Import Library** menu. This will insert one or more **#include** statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its **#include** statements from the top of your code.

There is a [list of libraries](http://arduino.cc/en/Reference/Libraries) in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch. See these [instructions for installing a third-party library](http://arduino.cc/en/Guide/Libraries).

To write your own library, see [this tutorial](http://arduino.cc/en/Hacking/LibraryTutorial).

**Third-Party Hardware**

Support for third-party hardware can be added to the **hardware** directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the **hardware** directory, then unzip the third-party platform into its own sub-directory. (Don't use "arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

For details on creating packages for third-party hardware, see the [platforms page](http://code.google.com/p/arduino/wiki/Platforms) on the Arduino Google Code developers site.

**Serial Monitor**

Displays serial data being sent from the Arduino board (USB or serial board). To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down that matches the rate passed to **Serial.begin** in your sketch. Note that on Mac or Linux, the Arduino board will reset (rerun your sketch from the beginning) when you connect with the serial monitor.

You can also talk to the board from Processing, Flash, MaxMSP, etc (see the [interfacing page](http://www.arduino.cc/playground/Main/Interfacing) for details).

**Preferences**

Some preferences can be set in the preferences dialog (found under the **Arduino** menu on the Mac, or **File** on Windows and Linux). The rest can be found in the preferences file, whose location is shown in the preference dialog.

**Language Support**

The Arduino 1.0.1 software environment has been translated into 30+ different languages. By default, the IDE loads in the language selected by your operating system. (Note: on Windows and possibly Linux, this is determined by the locale setting which controls currency and date formats, not by the language the operating system is displayed in.)

If you would like to change the language manually, start the Arduino software and open the **Preferences** window. Next to the **Editor Language** there is a dropdown menu of currently supported languages. Select your preferred language from the menu, and restart the software to use the selected language. If your preferred language is not supported, the IDE will default to English.

You can return Arduino to its default setting of selecting its language based on your operating system by selecting **System Default** from the **Editor Language** drop-down. This setting will take effect when you restart the Arduino software. Similarly, after changing your operating system's settings, you must restart the Arduino software to update it to the new default language.

**Boards**

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection you'll want to check it before burning the bootloader.

* *Arduino Uno*   
  An ATmega328 running at 16 MHz with auto-reset, using the optiboot bootloader (115200 baud, 0.5 KB).
* *Arduino Duemilanove w/ ATmega328*   
  An ATmega328 running at 16 MHz with auto-reset.
* *Arduino Diecimila or Duemilanove w/ ATmega168*   
  An ATmega168 running at 16 MHz with auto-reset. Compilation and upload is equivalent to Arduino NG or older w/ ATmega168, but the bootloader burned has a faster timeout (and blinks the pin 13 LED only once on reset).
* *Arduino Nano w/ ATmega328*   
  An ATmega328 running at 16 MHz with auto-reset. Has eight analog inputs.
* *Arduino Nano w/ ATmega168*   
  An ATmega168 running at 16 MHz with auto-reset. Compilation and upload is equivalent to Arduino NG or older w/ ATmega168, but the bootloader burned has a faster timeout (and blinks the pin 13 LED only once on reset). Has eight analog inputs.
* *Arduino Mega 2560 or Mega ADK*   
  An ATmega2560 running at 16 MHz with auto-reset, using an stk500v2 bootloader.
* *Arduino Mega (ATmega1280)*   
  An ATmega1280 running at 16 MHz with auto-reset.
* *Arduino Leonardo*   
  An ATmega32u4 running at 16 MHz with auto-reset.
* *Arduino Mini w/ ATmega328*  
  An ATmega328 running at 16 MHz with auto-reset, using the optiboot bootloader (115200 baud, 0.5 KB). Has eight analog inputs.
* *Arduino Mini w/ ATmega168*   
  Equivalent to Arduino NG or older w/ ATmega168 (i.e. an ATmega168 running at 16 MHz without auto-reset).
* *Arduino Ethernet*   
  Equivalent to Arduino UNO with an Ethernet shield.
* *Arduino Fio*   
  An ATmega328 running at 8 MHz with auto-reset. Equivalent to Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328.
* *Arduino BT w/ ATmega328*   
  ATmega328 running at 16 MHz. The bootloader burned (4 KB) includes codes to initialize the on-board bluetooth module.
* *Arduino BT w/ ATmega168*   
  ATmega168 running at 16 MHz. The bootloader burned includes codes to initialize the on-board bluetooth module.
* *LilyPad Arduino w/ ATmega328*   
  An ATmega328 running at 8 MHz (3.3V) with auto-reset. Equivalent to Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328.
* *LilyPad Arduino w/ ATmega168*   
  An ATmega168 running at 8 MHz. Compilation and upload is equivalent to the Arduino Pro or Pro Mini (8 MHz) w/ ATmega168. The bootloader burned, however, has a slower timeout (and blinks the pin 13 LED three times on reset) because the original versions of the LilyPad didn't support auto-reset. They also didn't include an external clock, so the burn bootloader command configures the fuses of ATmega168 for an internal 8 MHz clock.   
    
  If you have a recent version of the LilyPad, (w/ a 6-pin programming header), you'll want to select Arduino Pro or Pro Mini (8 MHz) w/ ATmega168 before burning the bootloader.
* *Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328*   
  An ATmega328 running at 16 MHz with auto-reset. Equivalent to Arduino Duemilanove or Nano w/ ATmega328.
* *Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168*   
  An ATmega168 running at 16 MHz with auto-reset. Equivalent to Arduino Diecimila, Duemilanove, or Nano w/ ATmega168.
* *Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328*   
  An ATmega328 running at 8 MHz (3.3V) with auto-reset. Equivalent to LilyPad Arduino w/ ATmega328.
* *Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168*   
  An ATmega168 running at 8 MHz (3.3V) with auto-reset.
* *Arduino NG or older w/ ATmega168*   
  An ATmega168 running at 16 MHz *without* auto-reset. Compilation and upload is equivalent to Arduino Diecimila or Duemilanove w/ ATmega168, but the bootloader burned has a slower timeout (and blinks the pin 13 LED three times on reset).
* *Arduino NG or older w/ ATmega8*   
  An ATmega8 running at 16 MHz *without* auto-reset.

For instructions on installing support for other boards, see [third-party hardware](http://arduino.cc/en/Guide/Environment?from=Tutorial.Bootloader#thirdpartyhardware) above. The text of the Arduino getting started guide is licensed under a [Creative Commons Attribution-ShareAlike 3.0 License](http://creativecommons.org/licenses/by-sa/3.0/). Code samples in the guide are released into the public domain.

[Bookmark and Share](http://www.addthis.com/bookmark.php?v=300&pubid=xa-50f052251629971d)

©Arduino | [Copyright Notice](http://arduino.cc/en/Main/CopyrightNotice) | [Edit Page](http://arduino.cc/en/Guide/Environment?action=edit) | [Page History](http://arduino.cc/en/Guide/Environment?action=diff) | [Printable View](http://arduino.cc/en/Guide/Environment?action=print) | [All Recent Site Changes](http://arduino.cc/en/Site/AllRecentChanges)

**Burning sketches to the Arduino board with an external programmer**

If you have an external programmer (e.g. an AVR-ISP, STK500, or [parallel programmer](http://arduino.cc/en/Hacking/ParallelProgrammer)), you can burn sketches to the Arduino board without using the bootloader. This allows you to use the full program space (flash) of the chip on the Arduino board. So with an ATmega168, you'll get 16 KB instead of 14 (on an ATmega8 you'll get 8 KB instead of 7). It also avoids the bootloader delay when you power or reset your board.

**Editing the Arduino preferences file**

To use an external programmer for all boards you need to make some small changes to the Arduino [preferences.txt](http://arduino.cc/en/Hacking/Preferences). Make sure to edit this file when the Arduino IDE is not running.

This preferences file is found in this folder:

\* /Users/<USERNAME>/Library/Arduino/preferences.txt (Mac)

\* c:\Documents and Settings\<USERNAME>\Application Data\Arduino\preferences.txt (Windows)

\* ~/.arduino/preferences.txt (Linux)

Note that when you upgrade Arduino, this folder may not be deleted. If you are having trouble, delete this folder. Upon launching Arduino a new file and preferences.txt will be created.

Change: upload.using from bootloader to the identifier of one of the programmers in hardware/programmers.txt (e.g. avrispmkii).

If you would rather use an external programmer for only an individual board, you can edit the boards.txt file in the hardware/ sub-directory of the Arduino application directory. Set the *board*.upload.using parameter to the identifier of one of the programmers in programmers.txt.

**Burning the sketch**

After you've made these changes, you can burn sketches to the board with the normal upload button or menu item in the Arduino environment. The board needs to be powered, but you don't have to press the reset button before uploading.

You must specify which Arduino board you are using, but you do not have to specify a serial port (on Mac at least).

**Going back to normal USB/serial upload**

To go back to uploading sketches over the USB or serial port without an external programmer, you'll need to set the upload.using preference in preferences.txt back to bootloader; or remove the *board*.upload.using parameter from boards.txt. You'll also need to burn the [bootloader](http://arduino.cc/en/Hacking/Bootloader) back onto the chip on the Arduino board.

**Burning Sketches to the Arduino Mini**

To burn sketches to the mini you must first make a cable to interface between your programmer and the mini board. Gian Pablo Vilamil has written a very good tutorial on [Bootloading the Mini](http://www.arduino.cc/en/Hacking/MiniBootloader). To burn sketches, follow the instructions on bootloading the mini.

Once you have made the proper cable, instead of choosing bootloading from the arduino menu, you can use the instructions on this page to burn sketches directly to the Mini.