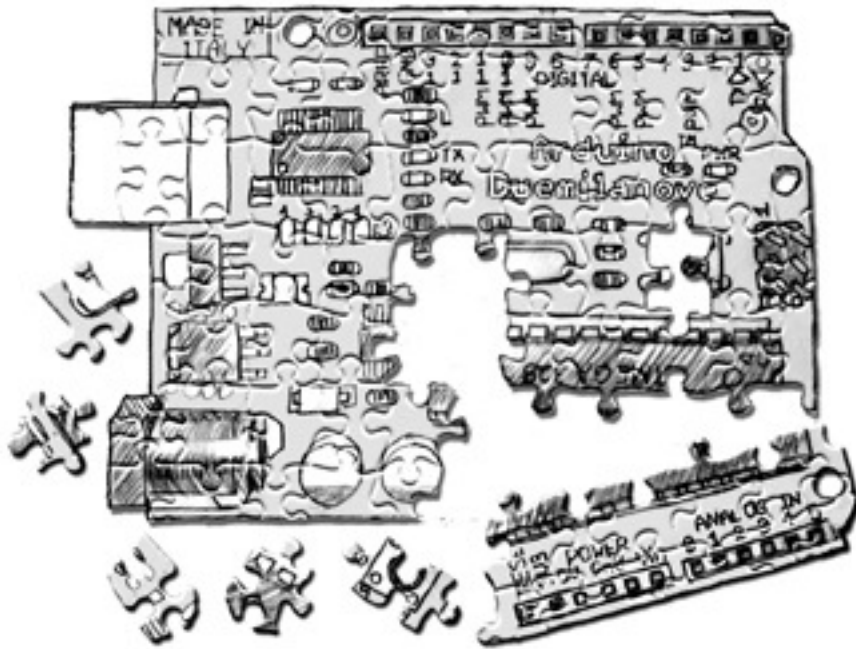


# **An Arduino Workshop – Excerpt includes chapters 1, 2, and 3.**

All illustrations are reduced from 300dpi  
in the text to 72 dpi for the excerpt to  
lower the file size



Joe Pardue - [SmileyMicros.com](http://SmileyMicros.com)



Copyright © 2010 by Joe Pardue, All rights reserved.  
Published by Smiley Micros

Smiley Micros  
5601 Timbercrest Trail  
Knoxville, TN 37909  
Email: [book@SmileyMicros.com](mailto:book@SmileyMicros.com)  
Web: <http://www.SmileyMicros.com>

ISBN 978-0-9766822-2-6

Products and services named in this book are trademarks or registered trademarks of their respective companies. In all instances where Smiley Micros is aware of a trademark claim, the product name appears in initial capital letters, in all capital letters, or in accordance with the vendor's capitalization preferences. Readers should contact the appropriate companies for complete information on trademarks and trademark registrations. All trademarks and registered trademarks in this book are the property of their respective holders.

No part of this book, except the programs and program listings, may be reproduced in any form, or stored in a database of retrieval system, or transmitted or distributed in any form, by any means, electronic, mechanical photocopying, recording, or otherwise, without the prior written permission of Smiley Micros or the author. The programs and program listings, or any portion of these, may be stored and executed in a computer system and may be incorporated into computer programs developed by the reader.

**NONE OF THE HARDWARE USED OR MENTIONED IN THIS BOOK IS GUARANTEED OR WARRANTED IN ANY WAY BY THE AUTHOR. THE MANUFACTURERS OR THE VENDORS THAT SHIPPED TO YOU MAY PROVIDE SOME COVERAGE, BUT THAT IS BETWEEN YOU AND THEM. NEITHER THE AUTHOR NOR SMILEY MICROS CAN PROVIDE ANY ASSISTANCE OR COMPENSATION RESULTING FROM PROBLEMS WITH THE HARDWARE.**

The information, computer programs, schematic diagrams, documentation, and other material in this book are provided "as is," without warranty of any kind, expressed or implied, including without limitation any warranty concerning the accuracy, adequacy or completeness of the material or the results obtained from the material or implied warranties. Including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. *Neither the publisher nor the author shall be responsible for any claims attributable to errors, omissions, or other inaccuracies in the material in this book. In no event shall the publisher or author be liable for direct, indirect, special, exemplar, incidental, or consequential damages in connection with, or arising out of, the construction, performance, or other use of the material contained herein. Including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any ay out of use, even if advised of the possibility of such damage. In no case shall liability be implied for blindness or sexual impotence resulting from reading this statement although the author suggests that if you did read all this then you really need to get a life.*

## Table of Contents:

Chapter 1: Introduction .....	13
So what is Arduino? .....	14
Smiley Micros Arduino Projects Kit.....	15
A Few Definitions.....	16
Genesis of Arduino.....	18
Processing.....	18
Wiring.....	18
Arduino.....	19
Smiley’s Workshop Nuts&Volts Articles .....	19
USB Serial Port .....	19
How is this book organized? .....	20
Prerequisites: .....	20
IMHO (In My Humble Opinion) is implied.....	21
Errors happen .....	21
“‘We really value your business, but all our lines are busy ...’”.....	21
Is Arduino the end or the beginning? .....	21
Where to get your Arduino.....	22
Chapter 2: Arduino Quick Start Guide.....	23
The Arduino Integrated Developers Environment .....	23
Select the Arduino Duemilanove with ATmega328 .....	26
Load the Blink Program, uhh... Sketch .....	27
Blink Source Code: .....	28
Verify the Sketch (Compile the Program).....	28
Uploading the sketch (program) to the Arduino board .....	30
Getting Help .....	30
Using the Arduino Language Reference .....	31
Using Internet Forums.....	33
Yes, you will get flamed .....	33
Chapter 3: Playing with software – Part 1 .....	35
How do we write a program? .....	35
How an Arduino Program is structured.....	36
Structure of the Arduino Blink example .....	37
Learning the programming language rules.....	38
An Arduino Light Switch.....	39
What do we want our program to do.....	39

Set up the hardware .....	40
Light_Switch - source code:.....	41
Some of the rules.....	43
Comments.....	43
Functions .....	43
Expressions, Statements, and Blocks .....	45
Flow Control .....	45
Chapter 4: Hardware Prototyping .....	47
Introduction to Breadboards.....	47
How a breadboard works.....	47
An Introduction to Schematics.....	50
Schematic Symbols .....	51
Using a breadboard with the Arduino .....	52
The Arduino Learning Platform (ALP).....	53
ALP Base.....	55
Arduino Learning Platform Storage Box .....	56
Chapter 5: Some Simple Projects.....	59
Digital Input and Output .....	59
DIP Switch and LEDs .....	59
The DIP Switch .....	60
DIP to LED Source Code:.....	61
Cylon Eyes the Arduino Way.....	62
Cylon Eyes 1 Source Code:.....	62
Output sound – piezo music .....	66
Sounds Components, Schematic, Layout.....	67
Tunes .....	68
Happy Birthday Source Code:.....	70
Analog .....	72
Using PWM to Fade an LED .....	72
Chapter 6: Playing with electricity.....	75
Electricity is dangerous (well, duh!) .....	75
Electric Measurements .....	76
Electric Potential Difference = Voltage .....	77
Electric Current = Amps .....	78
Electric Resistance = Ohms ( $\Omega$ ).....	79
Ohm’s Law .....	79
Circuits .....	80

Short Circuits.....	81
Electrifying Experiments.....	82
An Arduino Volt Meter .....	82
Installing the Arduino Volt Meter on your PC.....	83
Opening the Arduino in the Volt Meter .....	83
AVM_Test Source Code: .....	84
Voltage across resistance .....	84
Variable Resistance: the Potentiometer.....	88
LED Dimmer.....	93
Chapter 7: Playing with software – Part 2 .....	97
Microcontroller I/O Ports .....	97
DIPLED_With_Ports Source Code:.....	99
Redo Cylon Eyes using ports .....	100
Cylon Eyes 2 Source Code:.....	100
Operators .....	102
Some equals are more equal than others .....	104
There are exactly 10 types of people in the world. Those who understand binary and those who don't. ....	104
Bits .....	104
Bytes.....	105
Bitwise Operators.....	107
The TRUTH about numbers.....	107
ORing .....	108
ANDing .....	109
Setting and Clearing Bits.....	110
XORing .....	111
NOTing.....	111
Bitwise versus Boolean Operators .....	112
Launch_Control Source Code: .....	114
Shift Operators .....	115
Masks and Macros: Using Named Bits .....	116
CylonOptometry Source Code: .....	119
Chapter 8: Communicating with a PC .....	129
ASCII and the char data type .....	129
So what's the big deal about ASCII anyhow?.....	130
Sending an receiving numbers .....	130
Number_Commander Source Code: .....	132

Arduino Jukebox Source Code:.....	134
Getting real with serial input.....	139
ASCII_To_Integer Source Code: .....	141
Command Demonstration .....	143
Command_Demo Source Code:.....	144
Chapter 9: Sensors.....	149
Light Sensor .....	150
Light Sensor Components, Schematic, Layout .....	151
A Word or Two about Storing and Showing Sensor Data .....	152
A quick introduction to signed decimal numbers.....	152
Showing integer data as signed decimal fractions .....	153
Converting Centigrade to Fahrenheit .....	154
The LM35 Temperature Sensor .....	155
Temperature Sensor Components, Schematic, Layout .....	157
LM35_Temperature Source Code:.....	159
Infrared Object Detection.....	162
But is it Light?.....	162
Some Common Uses for IR Sensors .....	164
IR Reflective Object Sensor – the QRD1114 .....	165
Making IR visible.....	165
IR doesn't know black from white.....	166
Show invisible IR intensity with visible red LED.....	166
Tomato Soup Can Counter .....	170
Tomato Soup Can Counter Software .....	171
ReadPot Source Code:.....	171
Tomato_Soup_Can_Counter Source Code: .....	173
Chapter 10: Simple Motor Speed Control.....	177
Using external interrupts to detect edges .....	178
Edge Detect Interrupt Software.....	180
Edge_Detect_Interrupt Source Code:.....	180
Optical Isolation of voltage .....	182
Optical Isolation Component, Schematic, Layout .....	184
Optical Isolation Source Code.....	185
PWM_Test Source Code:.....	185
How our DC motor works.....	188
Diode to suppress voltage spikes .....	191
Powering the motor .....	192

Using PWM to control the motor speed.....	193
Building the breadboard circuit.....	194
Using an encoder wheel to measure the motor speed .....	197
Simple motor speed control with digital feedback.....	199
Simple_Motor_Speed_Control Source Code:.....	199
Now what?.....	203
Appendix 1: ASCII Table .....	205
Appendix 2: Decimal, Hexadecimal, and Binary.....	207
Index.....	209

## Table of Figures:

Figure 1: The Arduino Duemilanove .....	13
Figure 2: The Arduino Projects Kit.....	15
Figure 3: Arduino Projects Kit Parts List.....	16
Figure 4: Bogus Security Warning.....	23
Figure 5: cmd.exe window .....	24
Figure 6: Arduino desktop icon.....	24
Figure 7: The Arduino Integrated Developers Environment .....	25
Figure 8: Select the Duemilanove board .....	26
Figure 9: Selecting the Blink example .....	27
Figure 10: Verify the sketch.....	29
Figure 11: Compile.....	29
Figure 12: Done compiling .....	29
Figure 13: Upload to the I/O board .....	30
Figure 14: Help\Reference .....	31
Figure 15: Arduino Language Reference .....	32
Figure 16: pinMode function reference.....	33
Figure 17: Schematic – Pushbutton switch and LED.....	39
Figure 18: Drawing - pushbutton switch and LED .....	40
Figure 19: Breadboard front and back.....	48
Figure 20: Back with 5-position vertical and 50-position clips .....	48
Figure 21: A 5-position clip .....	49
Figure 22: Breadboard cross-section with LED, resistor, and wire .....	49
Figure 23: Layout Drawing and EAGLE Schematic for LED .....	51
Figure 24: Arduino pin-out schematic .....	52
Figure 25: The ALP base. ....	53
Figure 26: Slide it in the ALP box .....	54
Figure 27: Scruffy protective box .....	54
Figure 28: Construction site .....	55
Figure 29: Stick on Velcro squares .....	56
Figure 30: Start assembly .....	56
Figure 31: Glue the sides together as shown.....	57
Figure 32: Front before taping lid .....	57
Figure 33: DIP switch and LEDs drawing .....	59
Figure 34: 8-position DIP switch. ....	60
Figure 35: DIP switch and LED schematic.....	61

Figure 36: Piezo element layout.....	66
Figure 37: Piezo element illustration .....	67
Figure 38: Piezo element schematic.....	68
Figure 39: Note table.....	68
Figure 40: ‘c’ note waveform.....	69
Figure 41: PWM Fade .....	73
Figure 42: Learning the shocking truth about electricity. ....	75
Figure 43: Bucket voltage metaphor .....	77
Figure 44: Amp metaphor .....	78
Figure 45: Flow of conventional current.....	80
Figure 46: Arduino Volt Meter .....	82
Figure 47: Select your Arduino.....	83
Figure 48: Schematic of resistors in series.....	85
Figure 49: Layout of resistors in series .....	86
Figure 50: AVM measuring resistor divider .....	87
Figure 51: Potentiometer.....	88
Figure 52: Potentiometer metaphor and real circuit.....	89
Figure 53: The potentiometer wiper.....	91
Figure 54: Potentiometer to Arduino schematic .....	92
Figure 55 Arduino reads potentiometer .....	92
Figure 56: Pot, LED, and Arduino schematic .....	93
Figure 57: Pot, LED, and Arduino drawing.....	94
Figure 58: AVM AT 1.62.....	95
Figure 59: Mapping ATmega168/328 to Arduino pins.....	98
Figure 60: Arduino Pin9 is ATmega328 PortB Pin1 .....	99
Figure 61: DIP Switch Use.....	117
Figure 62: Select the Arduino IDE Serial Monitor .....	133
Figure 63: Number Commander in the Serial Monitor.....	134
Figure 64: Arduino IDE Serial Monitor ASCII_To_Integer.....	143
Figure 65: Command Demo .....	144
Figure 66: CdS light sensor layout.....	149
Figure 67: CdS light sensor.....	150
Figure 68: CdS light sensor schematic .....	151
Figure 69: Centigrade mapped to Fahrenheit.....	154
Figure 70: Temperature, voltage, and ADC ranges .....	156
Figure 71: LM35 temperature sensor .....	157
Figure 72: Temperature sensor schematic.....	158

Figure 73: Temperature sensor layout drawing.....	159
Figure 74: Output in Arduino Serial Monitor .....	161
Figure 75: Cute L'il Bunny has her last thought.....	162
Figure 76: Dog IR thermoscan .....	163
Figure 77: Digital camera 'sees' TV remote IR.....	164
Figure 78: QRD1114 Cross-section .....	165
Figure 79: QRD1114 Reflective Object Sensor .....	166
Figure 80: Bending the QRD1114 legs .....	167
Figure 81: Object Detector Schematic .....	168
Figure 82: IR Object Detector Layout.....	168
Figure 83: IR Object Detector Photo.....	169
Figure 84: Tomato soup can counter prototype .....	170
Figure 85: ADC readings of finger movements .....	172
Figure 86: Can count on Arduino serial monitor. ....	175
Figure 87: Simple Motor Speed Control .....	177
Figure 88: Edge detection schematic .....	180
Figure 89: Edge Detect Interrupt counter.....	181
Figure 90: 4N25 Optically Coupled Isolator.....	182
Figure 91: QRD1114 and 4N25 schematic symbols.....	183
Figure 92: Optoisolation Test Layout .....	184
Figure 93: Optoisolator Test Circuit .....	185
Figure 94: DC Motor Principles.....	188
Figure 95: DC motor dismantled.....	190
Figure 96: Diode.....	191
Figure 97: PWMs on oscilloscope .....	193
Figure 98: DC Motor.....	195
Figure 99: Power Transistor .....	196
Figure 100: Motor Speed Control Schematic.....	196
Figure 101: Motor Speed Control Layout.....	197
Figure 102: Encoder Wheel.....	198
Figure 103: Program Serial I/O .....	199



## Chapter 1: Introduction

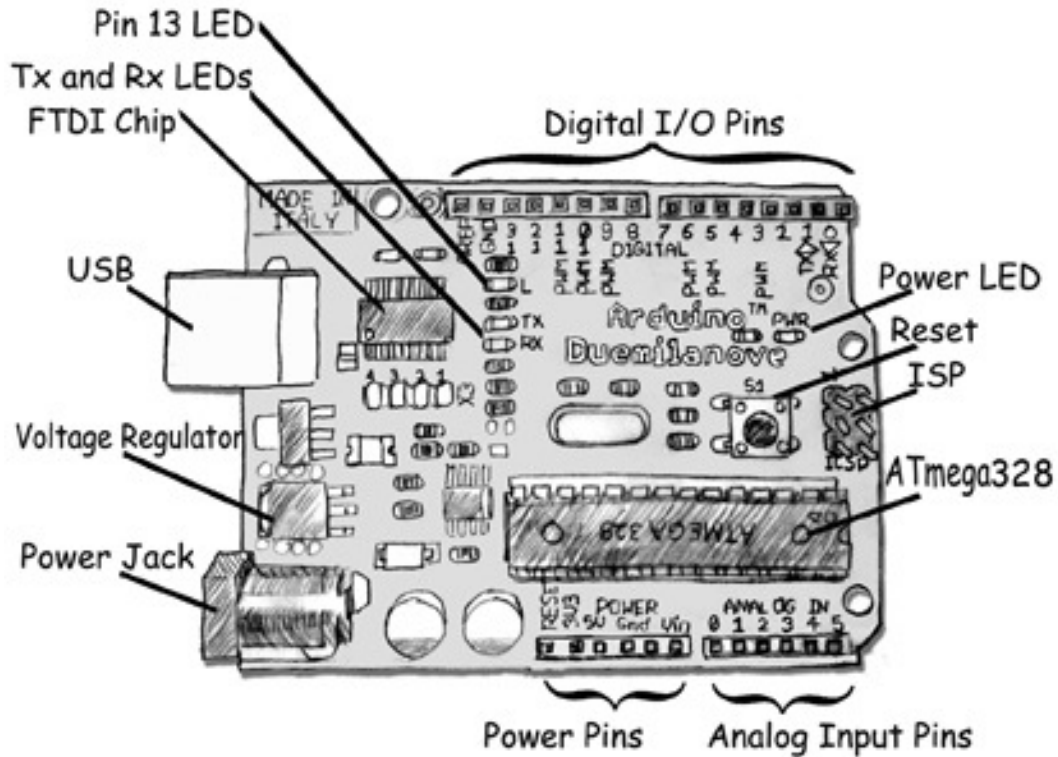


Figure 1: The Arduino Duemilanove

**Work?** I put ‘workshop’ in the title so that you would know that you are going to get your hands dirty. This is not a book to feed ideas into your brain while you relax on a deck chair drinking Piña Coladas. It is a book where you spread your Smiley Micros Arduino Projects Kit out on a table next to a PC and build stuff to help you learn how to **play** with microcontrollers and electrical hardware.

**Play?** Yes it is not just work, and the real emphasis here is play. If it ain’t fun, why do it?

One of the really cool things about the Arduino is that you can start playing with it without understanding much of what you are doing. We will begin with this concept in mind and play with the Arduino and some of the stuff in the Projects Kit. Later, we will begin to learn about the details of how the software and hardware work. Be aware, however, that the presentation of topics may not seem entirely logical, because, well... it isn't particularly logical. I organized this book thinking about how I might present various topics to a novice in a series of hands-on workshops. This means that you may first learn about a certain software operator in the midst of a discussion about measuring temperature. You will find some of what is presented is intuitively obvious and other stuff may cause you WTF moments.

### ***So what is Arduino?***

Massimo Banzi begins his book, *Getting Started with Arduino*:

“A few years ago I was given the very interesting challenge: teach designers the bare minimum in electronics so that they could build interactive prototypes of the objects they were designing.” He summarizes his philosophy of ‘learning by tinkering’ with a quote from [www.exploratorium.edu/tinkering](http://www.exploratorium.edu/tinkering): “Tinkering is what happens when you try something you don’t quite know how to do, guided by whim, imagination, and curiosity. When you tinker, there are no instructions – but there are also no failures, no right or wrong ways of doing things. It’s about figuring out how things work and reworking them.”

Arduino provides a great toolset for designers, tinkers, and even some of us surly old engineers who sometimes just want to play with an idea. The genius of Arduino is that it provides just enough access to get specific tasks done without exposing the underlying complexities that can be truly daunting for folks new at this stuff.

## *Smiley Micros Arduino Projects Kit*

You can get the Arduino Projects Kit from [www.smileymicros.com](http://www.smileymicros.com).

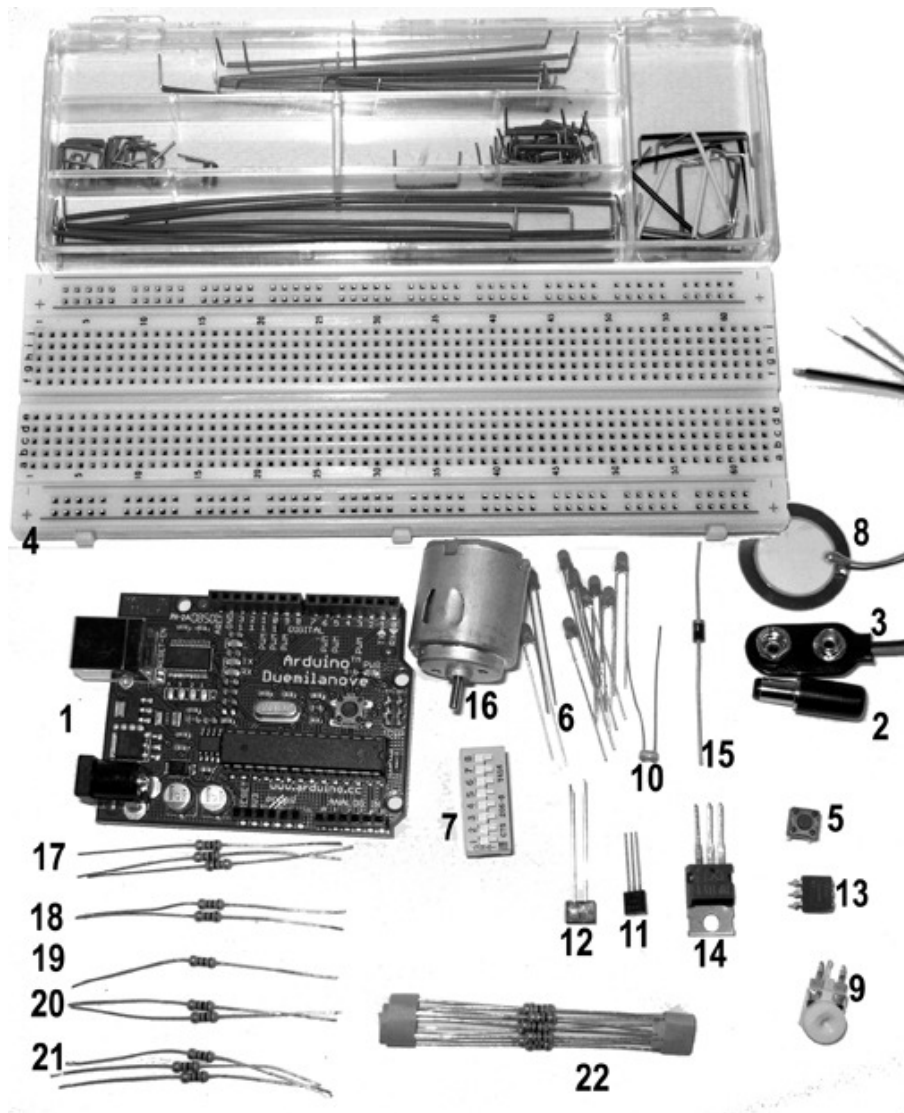


Figure 2: The Arduino Projects Kit

Part #	Arduino Projects Kit
1	Arduino Duemilanove (ATmega328)
2	DC Plug 2.1mm(ID)5.5(OD) 9.5mm
3	9V Battery Snap
4	Breadboard + Jumper wire kit
5	Push button
6	10 - LED Red 3mm
7	DIP Switch SPST 8pos
8	Piezo element
9	Potentiometer 10K
10	CdS Light Sensor
11	LM35 Temperature Sensor
12	IR Reflector QRD1114
13	Optoisolator 4n28
14	Power Transistor TIP115
15	Diode 1N4001
16	Motor DC 6-12V
17	3 -100 $\Omega$ Resistor
18	2 - 150 $\Omega$ Resistor
19	1- 270 $\Omega$ Resistor
20	1 - 2.2k $\Omega$ Resistor
21	3- 10k $\Omega$ Resistor
22	10 - 1k $\Omega$ Resistor

Figure 3: Arduino Projects Kit Parts List

### A Few Definitions.

**Sensor** – a computer-controlled mechanism that measures or detects events in the real world such as motion, light, or voltage. For example: an IR optical sensor used to detect the motion of a computer mouse.

**Actuator** – a computer controlled mechanism that causes a device to be turned on or off, adjusted or moved. For example: the devices that spin a DVD, move the sensing head, and turn the read/write laser on/off.

**Micro (microcontoller)** – a single integrated circuit that contains a computer, memory, and peripherals used with sensors and actuators to provide **control** of

## Chapter 1: Introduction

real-world events. For example: an Atmel AVR microcontroller such as the ATmega328 can be used to sense button presses, motor rotation, and temperature and use that input to control the voltage to the cavity magnetron (makes the microwaves) in a microwave oven.

**Real World** – for our purposes, this is everything outside our micro and associated sensors and actuators, though to be completely honest, I gave up on trying to define ‘real world’ years ago.

**Physical Computing (Embedded Systems)** – a combination of micro(s), sensor(s), and actuator(s) designed for some specific control function and ‘embedded’ into a specific device, usually requiring little human input. An example would be the air/fuel mixture control system in an automobile engine. This term is often used in contrast to a ‘general-purpose computer’ such the PC or Mac, which can do a lot of different things and are designed for intense human interaction.

**Sketch (program)** – a sequence of human readable text statements following the rules of a programming language that can be converted and uploaded to a micro providing instructions as to how it will make decisions about using its sensors and actuators.

**Verify (compile)** – the process of converting a sketch to instructions suitable for the micro. The process provides feedback if the conversion cannot be completed due to errors.

While using Arduino is relatively simple, it is built on top of some things that can quickly become complex if you dig a little below the surface.

### **Arduino Hardware Resources (see Figure 1)**

- 14 Digital I/O pins, six of which can be configured as analog output (PWM) pins
- 6 Analog Input pins
- Pins are on female sockets that are easy to wire to a breadboard
- USB serial port with +5V bus power through a PTC fuse.
- Serial transmission and reception LEDs.
- May be switched to regulated external power socket.
- ATmega328 with a pre-programmed bootloader.
- ISP (In-System Programming) header
- Reset button

## *Genesis of Arduino*

Verily in the olden days, artists wailed for lack of entry to the new land. And they were heard by great prophets who showed them a path: Processing. And lo, it came to pass that Processing begat Wiring which begat Arduino that, the time being ripe, begat a whole nation of xxx-duino derivatives. This dear brethren is the story:

### **Processing**

You have probably seen some of the cool (or annoying depending on your viewpoint) graphic Java applets that bless (or infest) Internet browsers. But things like those ‘too darn cute’ rotating bouncing cubes with different pictures or messages on each face did show that you could use Java to build some very interesting graphic things, and artists took notice. The only problem is that artists tend not to be programmers and vice versa. Casey Reas and Benjamin Fry while in the Aesthetics and Computation Group at the MIT Media Lab took note of the potential for Java to be used by artists to do artsy stuff and decided to create a simpler interface that would be more artist friendly. They developed Processing: a programming language, development environment, and online community [[www.processing.org](http://www.processing.org)]. The syntax looks a lot like C (but then so does Java if you squint). The Arduino Integrated Development Environment (IDE) is derived from the Processing IDE.

Remember that Processing was designed for artists. It substitutes ‘sketch’ for ‘program’ and ‘verify’ for ‘compile’. If you are at all interested in art, be sure and go to their web site and click on the exhibition tab.

### **Wiring**

Wiring was started by Hernando Barragán at the Interaction Design Institute Ivrea in Italy and builds on Processing to provide an ‘electronics prototyping input/output board for exploring arts and tangible media.’ The Arduino folks smash this mouthful into ‘physical computing’ and I’m tempted to redub it ‘playing with electrons’ because it makes things so much easier than many earlier systems that the concept of ‘play’ is not out of the question. The Wiring board is based on the ATmega128 and the language has libraries built around that device’s resources and tasks you might like to do with them (like blinking LEDs or monitoring a microphone). The good news is that this is a feature rich platform

and the bad news is that all those riches cost money. So enter the Arduino designed to be affordable for students with the cost goal being the same as going out and getting a pizza.

### **Arduino**

Arduino is built directly out of Wiring except that it uses a more affordable board, and has a very active online community. Massimo Banzi ([www.tinker.it](http://www.tinker.it)) thought up the name Arduino and, if the story is true, it was christened five minutes before the first board was to go into production. Massimo was on the phone with the PCB fabricator who wanted a name for the board and Massimo suggested the name of a bar that some of the developers once frequented. And while the name may be somewhat accidental, it is a masterstroke of Google-ability especially compared to the names Processing or Wiring (try Goolge-ing the three and see what I mean). Since this is all open-source, it didn't take long for lots of folks to make their own versions of the board and use the –'duino' in the name as if it was a valid suffix. This apparently irritates some Italians who note, for instance, that the board name Sanguino would be what an Italian would shout if he was bleeding. There was a bit of debate on the Arduino forum about PCB designs perhaps straying too far from the original concept and making it difficult to provide support so several of the originators copyrighted the name and allow its use for boards that pass muster. Naturally a conspiracy theory rebellion occurred and some folks mounted their high horses and came up with the name Freeduino as a 'free' alternative ([www.freeduino.org](http://www.freeduino.org)).

### **Smiley's Workshop Nuts&Volts Articles**

This book is based very loosely on a series of articles in Nuts&Volts magazine: Smiley's Workshop parts 9 through 17. Those articles have been reorganized, rewritten, and expanded for this book so expect some overlap if you read them, but be sure and note that those articles aren't a substitute for this book.

### **USB Serial Port**

By the way, the USB serial port on the Arduino uses the FTDI FT232R chip which was discussed in detail in the article "The Serial Port is Dead, Long Live the Serial Port" by yours truly in the June 2008 issue of Nuts&Volts (you can get a copy on [www.smileymicros.com](http://www.smileymicros.com)). And if you come to really like that chip you might want to get the book "*Virtual Serial Programming Cookbook*" (also by yours truly) and associated projects kit from the Smiley Micros website.

## ***How is this book organized?***

I find it very difficult to organize information to help folks understand microcontroller hardware and software in a reasonably logical sequence for both hardware and software at the same time. It seems that I have to choose either hardware or software as the central theme and then let the other be a supporting mishmash. In my *C Programming for Microcontrollers* book, I chose the programming language as the guide and the hardware got slapped in, to support software examples. In this book the hardware is the guide and the software is crammed in as needed.

### **Prerequisites:**

1. You have access to a PC with a USB port and a high-speed Internet connection. I personally think that a good Internet connection is the best learning tool ever invented, so get one.
2. You know simple math concepts. If you are given the equation  $X = Y/2$  and that Y is equal to 10 and you can tell me what X is, then you should do just fine.
3. Don't even start unless:
  - a. You have near saint-like patience. Even though this stuff is as easy as I can make it, and loads of fun for most folks, it is still hard and will take lots of time and work.
  - b. You love puzzles. You will rarely get one of the projects working first time without having to puzzle out something.

I include the #3 prerequisite since I am a frequent and somewhat surly contributor to the forum on AVRfreaks and several times a week I read a thread posted by someone whose problem IMHO (see next page) is that the person lacks patience or doesn't like solving puzzles. Such folks really should be doing something else with their time. I love this stuff, and I have to assume that if you are reading this, then you are one of the folks who think you might love it also. But if your first instinct on encountering a problem is to go on an Internet forum and plead for help – do yourself a favor and don't even start.

## ***IMHO (In My Humble Opinion) is implied***

I get a lot of really nice compliments for my writing but occasionally someone seems to think that I ought to be burned at the stake for some heresy I dared speak. I have fun writing and occasionally I'll be flip with an issue that turns out to be someone else's sacred cow. When I realize that I may be moving into a sensitive topic, I try to remember to insert IMHO to remind folks that I haven't put myself forward as some kind of Guru-Swami-Know-It-All, but just an ordinary smuck having fun and trying to share the fun with others as best I can.

### **Errors happen**

Also I make mistakes (lots of mistakes), and as much as I'd like to say I'm sorry, I won't because I'd be lying. The only people who don't make mistakes are the ones who don't do anything. I suggest that you look on [www.smileymicros.com](http://www.smileymicros.com) at the web page for this book and review the Errata before reading further. If you find an error that is not in the Errata, then please notify me at: [error@smileymicros.com](mailto:error@smileymicros.com).

### **“We really value your business, but all our lines are busy ...”**

As much as I'd like to answer questions from everybody who reads this book, I simply don't have the time. Probably more import is that I think of this work as being part of a community effort. If you have a question, then it is likely that other folks will have the same question, so it is far better that the question gets asked and answered in a public forum such as the ones on [www.arduino.cc](http://www.arduino.cc) or [www.avrfreaks.net](http://www.avrfreaks.net) so that everyone has access to the answer. If I see that a question gets asked repeatedly, I'll post a FAQ on the Smiley Micros web page associated with this book. Be sure and mention the book in the thread title to increase the chances I'll see it.

## ***Is Arduino the end or the beginning?***

One thing to note is that the book goes further than many folks may consider standard for an Arduino workshop, and by the end we will be moving well beyond the Arduino simplifications and into topics that I think will help you advance in your study of how to use microcontrollers in a more general sense.

## Chapter 1: Introduction

I suspect most folks who might describe themselves as artists and mainly want to do things like blink LEDs, read switches and spin motors will be more than happy to put up with Chapters 1 to 6, and then start skimming the text and copying projects, since it starts getting more detailed from Chapter 7 onward. I also suspect that some folks who are thinking about more advanced work, like going into a University level program in Mechatronics or EE will want to go all the way to the end of the book to get a leg up on the work they will be doing later.

The Arduino fills the stated goal by enabling designers to build interactive prototypes with the bare minimum of electronic knowledge - and actually exceeds it in many ways. And while it is an admirable goal, it is not my goal. My goal is to help folks get started down this fascinating path and to me the Arduino is a good start, but by no means an end point unto itself.

This isn't a linear prescribed program leading to any kind of certification, grade, or degree so you are the one who has to decide what is enough and what is too much.

### ***Where to get your Arduino***

People who fret about the cost of buying the original Arduino from the folks who designed it versus saving a few bucks buying a knock-off should do some soul searching over their priorities. I have no financial arrangements with the manufacturers of the original Arduino (the one with the picture of Italy on it) but I use it in my Arduino Projects Kit and recommend it as one way to give back to the community that created it in the first place.

## Chapter 2: Arduino Quick Start Guide

### The Arduino Integrated Developers Environment

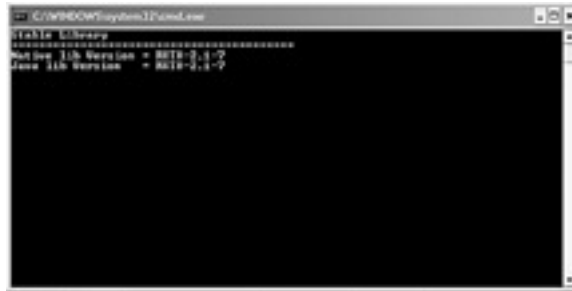
You can download the Arduino software from [www.arduino.cc](http://www.arduino.cc). This book is based on release 0015 working on a Windows XP PC (also tested with Vista Home Basic).

The Arduino folks seem to be posting new releases several times a year, so there is a possibility that the latest version available will have some changes that are not compatible with the release used in this book. If that turns out to be the case you can get release 0015 from my website.

After downloading to your directory, you can go ahead and click on `..\arduino-0015\Arduino.exe` and see what happens; it might run on your system. But in order to run the Processing IDE on my Windows XP machine I have to click on the `..\arduino-0015\run.bat` file. Windows doesn't like that so it displays a security warning (Figure 4: Bogus Security Warning). But to heck with Windows and its so-called security (like they would know security), just click 'Run'.



Figure 4: Bogus Security Warning



**Figure 5: cmd.exe window**

Which will open a 'cmd' window (Figure 5: cmd.exe window) and in a little while Windows will run Java for you and display the Processing IDE configured for the Arduino (Figure 7: The Arduino Integrated Developers Environment).

In Vista Home Basic, you only need to click on the Arduino icon (Figure 6: Arduino desktop icon). Don't be alarmed when your screen goes black for a moment, this is Java doing something weird. It will happen again when you close Arduino.



**Figure 6: Arduino desktop icon**

If your system doesn't fit any of these instructions, just go to [www.arduino.cc](http://www.arduino.cc) and download the latest and greatest version and follow their instructions.

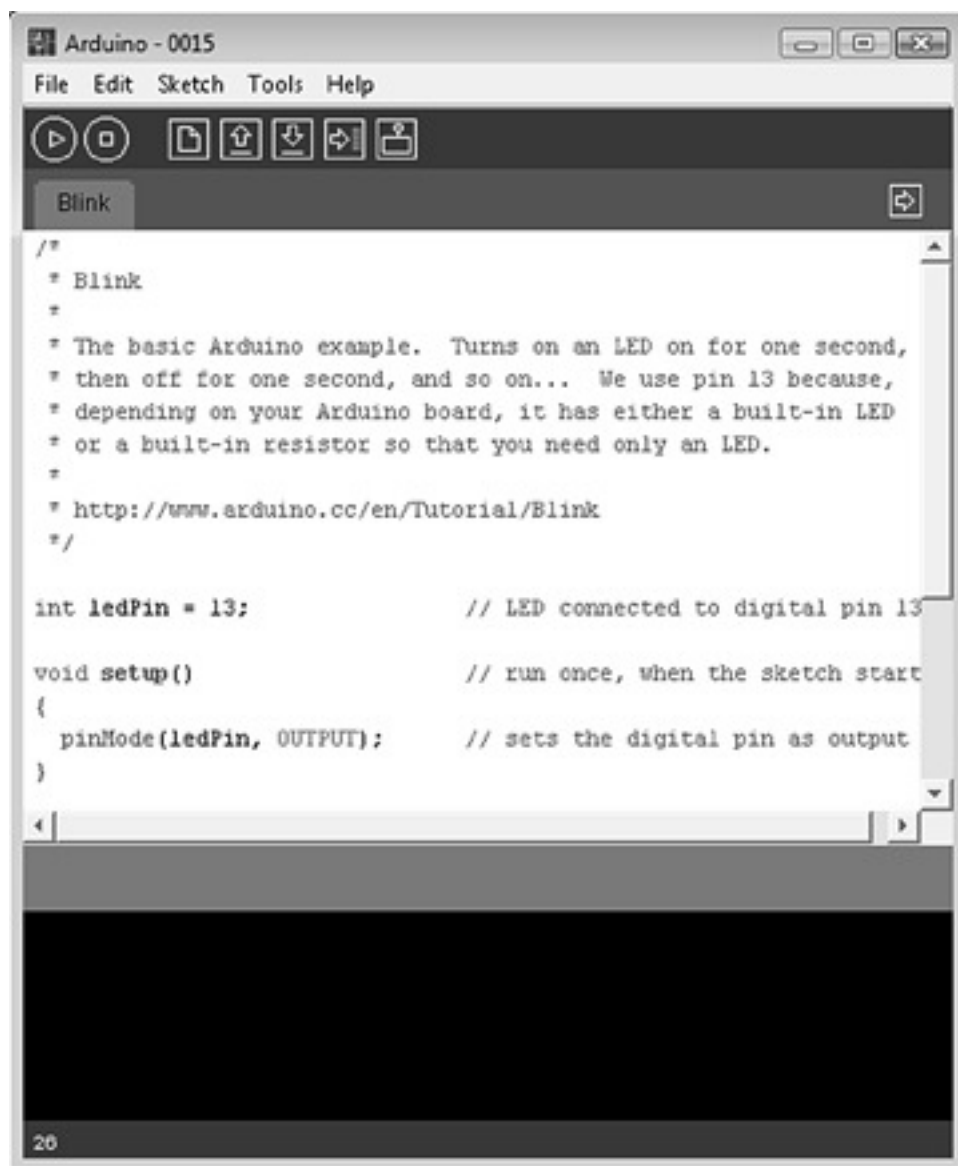


Figure 7: The Arduino Integrated Developers Environment

## Select the Arduino Duemilanove with ATmega328

Our Arduino Projects Kits have caused a little confusion because it uses the Duemilanove (Italian for 2009) with the ATmega328 (double the memory at the same price) instead of the older ATmega168. These processors have 32 or 16-kilo bytes of memory respectively so they require a different setting in the Arduino IDE. For our board, open the Tools/Boards menu and select the Arduino w/ ATmega328. Also note that the bootloader\* runs at 57600 baud, which is faster than the older Arduino bootloader and seems to be confusing some folks on the Arduino forum, so be careful.

Open the 'Tools' menu item, select 'Board', and then 'Arduino Duemilanove w/ATmega328' as shown in Figure 8: Select the Duemilanove board.

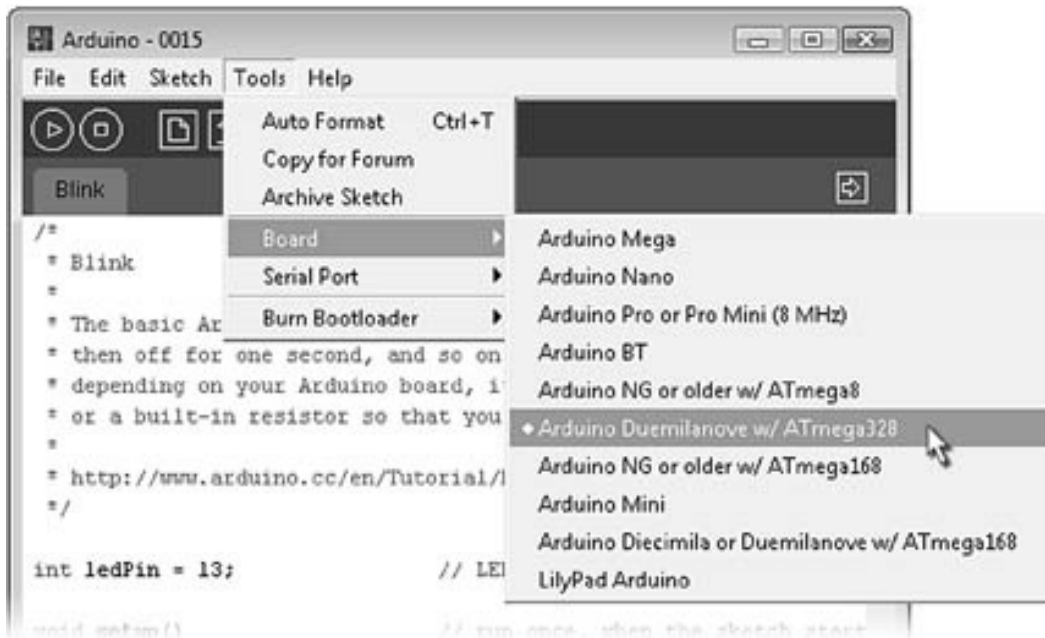


Figure 8: Select the Duemilanove board

\*A bootloader is software on the microcontroller that is used to upload application programs from the PC to the microcontroller.

## Load the Blink Program, uhh... Sketch

The Arduino Duemilanove has a built in LED with a resistor attached to I/O pin13 (see Figure 1: The Arduino Duemilanove), so we will start with a program (okay, sketch!) to blink that LED.

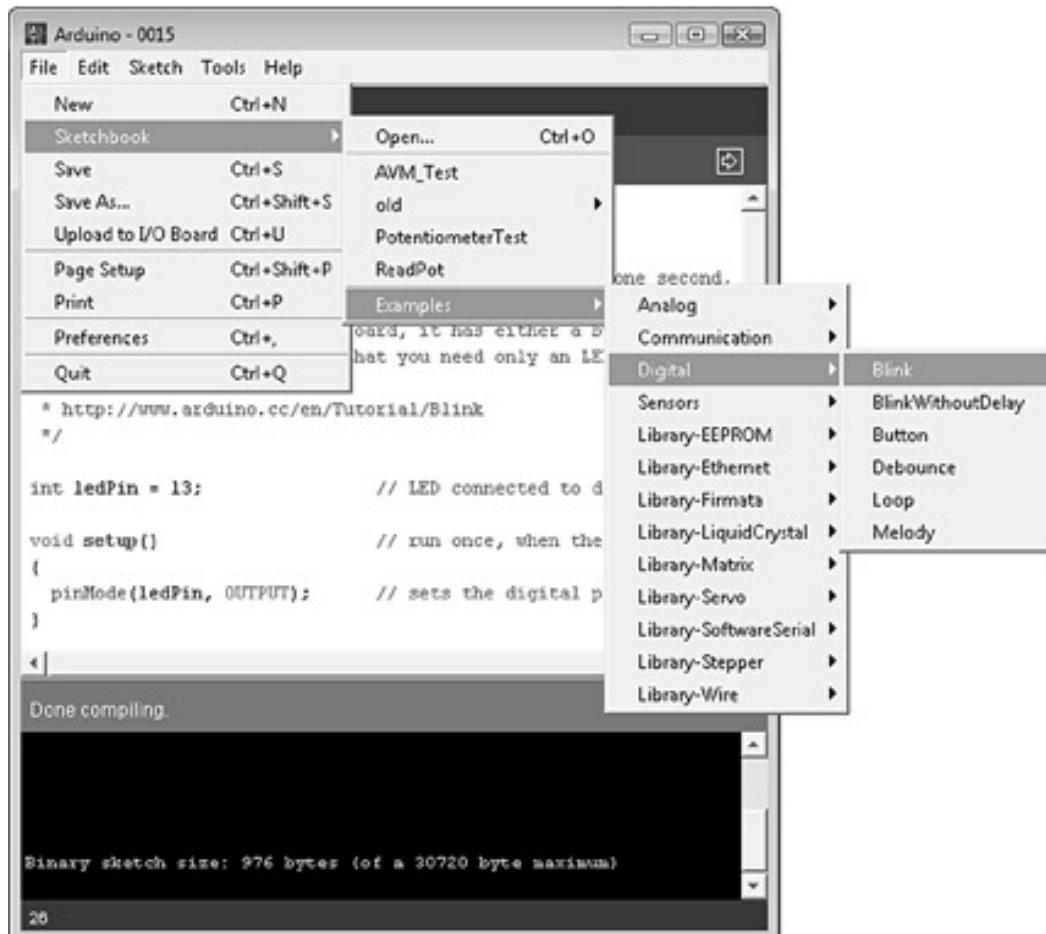


Figure 9: Selecting the Blink example

The following sketch was copied from the Arduino-0015 menu: File/Sketchbook/Examples/Digital/Blink (Figure 9: Selecting the Blink example) and modified to better fit in the book.

### Blink Source Code:

```
/*
 * Blink
 *
 * The basic Arduino example.
 * Turns on an LED on for one second,
 * then off for one second, and so on...
 * We use pin 13 because, depending on your
 * Arduino board, it has either a built-in
 * LED or a built-in resistor so that you
 * need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */
// LED connected to digital pin 13
int ledPin = 13;

// run once, when the sketch starts
void setup()
{
  // sets the digital pin as output
  pinMode(ledPin, OUTPUT);
}

// run over and over again
void loop()
{
  // sets the LED on
  digitalWrite(ledPin, HIGH);
  delay(1000); // waits for a second

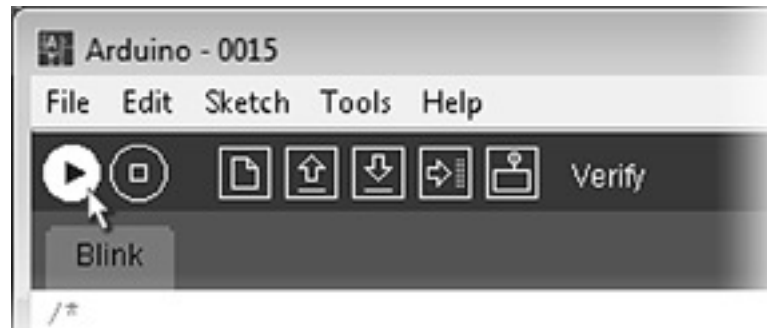
  // sets the LED off
  digitalWrite(ledPin, LOW);
  delay(1000); // waits for a second
}
```

And, no, you aren't expected at this point to understand any of the software. We will discuss those details later, but let's first run the code and blink that LED.

### Verify the Sketch (Compile the Program).

The Arduino folks also decided to call compile 'verify'. Instead of 'compiling the program' we are verifying the sketch'. [Artists... go figure] When you click on the button shown in Figure 10: Verify the sketch, the IDE uses the WinAVR

toolset for the GCC C compiler, and if everything goes okay – meaning it compiles without error, you don't have to know what WinAVR is, how it works, or even that it is there. We'll look at the situation involving errors later.



**Figure 10: Verify the sketch**

After you click verify you will notice some activity at the bottom of the Processing window. Figure 11: Compile shows the bottom block if the IDE you will see 'Compiling'.



**Figure 11: Compile**



**Figure 12: Done compiling**

And please don't ask me why they call it 'verify' at the top of the IDE, but 'compile' at the bottom. Hopefully you got the message: "Binary sketch size: 976 bytes (of a 30720 byte maximum" meaning everything is copasetic. If you wonder why there are only a maximum of 30720 bytes available when the ATmega328 has 32768 bytes of memory, it is because 2048 bytes are set aside for the bootloader.]

## Uploading the sketch (program) to the Arduino board

Next we will 'Upload to the I/O board' as shown in Figure 13: Upload to the I/O board, meaning we will secretly use AVRDUDE to send the hex code using the USB serial port to the bootloader on ATmega328 that will write it to memory. Though AVRDUDE is kept hidden, if you are curious, check out: <http://savannah.nongnu.org/projects/avrdude/>.

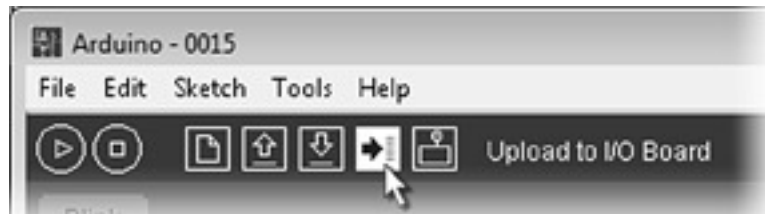


Figure 13: Upload to the I/O board

When you run this be sure and look at the TxD and RxD LEDs on the board right after you click the button so you can see them blinking rapidly in time to the flow of data between the Arduino board and the Arduino IDE. Sometimes it just helps to make things visible.

And after it loads, the Pin 13 LED should be blinking on and off once per second.

## Getting Help

Before we get too deep into the Arduino, we should look at what – IMHO - is the best feature of the whole Arduino way of doing things: the tools that can help you understand and use it. There are two main tools. First is the website

[www.arduino.cc](http://www.arduino.cc) - go there now and look around a bit. There are lots of things there that you may eventually find helpful. The second thing is the Help menu in the Arduino IDE. This item is seriously helpful. Click on it and browse around a bit to get a feel for what is there.

## Using the Arduino Language Reference

We have already used some Arduino software without explaining how it works. You may remember in the Blink example that we had this line:

```
pinMode(ledPin, OUTPUT);
```

The `pinMode()` function is part of the Arduino language and you can find details about it by clicking on the Help\Reference item as shown in Figure 14: Help\Reference.



Figure 14: Help\Reference

This will cause the Arduino Language Reference to open in a browser as shown in Figure 15: Arduino Language Reference.



Figure 15: Arduino Language Reference

In the right column under Functions - Digital I/O you will see `pinMode(pin,mode)`. Click on that and you get the page shown in Figure 16: `pinMode` function reference.



**Figure 16: pinMode function reference**

The discussions in this text should help familiarize you with many of the concepts, but the Help menu item is the ultimate resource.

### Using Internet Forums

Another great thing about the Arduino is the forum on their website where you can ask questions. But please before you go to the forum and ask for help, make sure you have read all the introductory materials, tried to find the answer to your question in the Arduino Language Reference, and thoroughly Googled your problem. And your life will be considerably easier if before you start using any Internet forum you read: <http://www.catb.org/~esr/faqs/smart-questions.html>. Once you've exhausted all available resources, then please don't hesitate to ask your question on the forum.

### Yes, you will get flamed

You will find many very helpful folks on the Internet, and you'll find some that are just plain bat-poop crazy and evil. I am not referring to the Arduino forum in

particular, since it seems to be a friendly place, especially when compared to some forums, but I just want to warn you that you need to disconnect your ego before using this resource. I have several friends who are robot enthusiasts who all agreed that they won't use my favorite forum (not the Arduino forum, and I won't name names here) because they are afraid of getting flamed. I had to bite my tongue to keep from saying, "Well, just hitch up your big-girl panties and get used to it. You want help on the Internet? You might get flamed and you might get help. But you sure as heck won't get help trembling in fear at the keyboard." There are folks on every forum who like to flame people. I even get so exasperated that I will occasionally get out my flame-thrower. But the thing to remember is that nobody actually knows you. You are no more a real person on the Internet, to some people, than a random character in a video game. Google 'Internet disinhibition' if this topic really interests you, but the take-home point is that some folks on Internet forums can be vicious while other's are nearly saints in there helpfulness, but none of it is aimed at you personally.

## Chapter 3: Playing with software – Part 1

### *How do we write a program?*

In the last chapter, we started playing with the Arduino and defined some terms. We defined a sketch (program) as a sequence of human readable text statements following the rules of a programming language that can be converted and uploaded to a micro providing instructions as to how it will make decisions about using sensors and actuators. While you will see the term ‘sketch’ frequently in Arduino discussions, from this point forward I will be using ‘program’ for that ‘human readable text’.

Since this discussion is novice oriented, I’m going to repeat some stuff so bear with me. Software may be thought of as the instructions that tell a micro how to do things. Telling a computer what to do is called programming. We write a software program using tools that allow us to provide these computer instructions in some human comprehensible form, like text files that use words and symbols available on a keyboard. Our text file (source code) must follow a set of rules that are provided for a specific programming language, and then we must submit that file to some compiler software that will then do one of two things: compile it or spit it back in your face. The spitting part is accompanied with a list of complaints that provide a guide to making the source file more palatable. There are two kinds of complaints: warnings and errors. Errors will prevent the code from compiling, but warnings won’t stop the compilation process, since the compiler will assume that you might just know what you are doing in the part that it is warning you about. You might be tempted to bypass warnings, but in general that is a very foolish thing to do.

If you didn’t make any mistakes in the text file, the compiler will translate your source code into instructions that the computer can use. If you did make a mistake, the compiler will try to give you a hint of the problem. Sometimes these hints are useful; sometimes they are not.

So a program is some text that follows some rules and can be converted into something that we can feed the micro that it can use to make decisions. Text, rules, and decisions – we learn the rules, then we write the text, then the micro makes decisions. This is the sequence:

1. We decide what we want the micro to do.
2. We think about how our programming language rules might be used to get the micro to do our bidding.
3. We write a program using the text editor in the Arduino IDE (Integrated Development Environment).
4. The Arduino IDE sends that text to a compiler that checks our text against some rules and then builds something that can be uploaded to the micro. The micro accepts the upload and starts making decisions about how the Arduino board uses the hardware.

In the real world we make mistakes at each point in the sequence and we have to rework each step (debugging) until eventually we get it right.

Following this list means that in order for us to write a program we are going to have to learn two things. What are the rules we can use and what are the decisions the micro can make? Fortunately we don't have to learn it all at once before we get started playing with these things. We will learn bits and pieces in small steps.

### ***How an Arduino Program is structured***

An Arduino program is structured in four parts.

**FIRST:** Begin with some comments about the program

**SECOND:** List variables and constants that all the functions may use. Variables are names for memory locations that a computer can use to store information that might change. Constants are numbers that won't change.

**THIRD:** Run the setup() function to prepare stuff. This is where you perform tasks that you want done once at the beginning of your program

```
void setup()  
{  
  // do things once at the start of the program  
}
```

**FOURTH:** Run the loop() function. This is where you run things in a sequence from the top of the loop to the bottom of the loop, then you start over again at the top, looping until the machine gets turned off

```
void loop()
{
  // Do the first thing

  // Do the second thing

  // Do any number of things

  // Do the last thing in the list

  // Go back to the beginning of this list
}
```

### **Structure of the Arduino Blink example**

I have taken the Blink program from the Arduino Examples and added the lines showing each of the four sections.

**FIRST:** Opening comments:

```
/*
 * Blink
 *
 * The basic Arduino example. Turns on an LED on for one second,
 * then off for one second, and so on... We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */
```

**SECOND:** List variables

```
int ledPin = 13; // LED connected to digital pin 13
```

**THIRD:** Things to do once at the beginning

```
void setup() // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}
```

### **FOURTH:** Do things in a loop

```
void loop()                                // run over and over again
{
  digitalWrite(ledPin, HIGH);             // sets the LED on
  delay(1000);                             // waits for a second
  digitalWrite(ledPin, LOW);              // sets the LED off
  delay(1000);                             // waits for a second
}
```

In the Quick Start Guide section we saw how to write a program in the Arduino IDE, compile it, and upload it to the Arduino board. You can locate this program in the Arduino IDE menu:

File\Sketchbook\Examples\Digital\Blink.

Click on the Blink then follow the instructions from the Quick Start Guide to compile and upload.

### ***Learning the programming language rules***

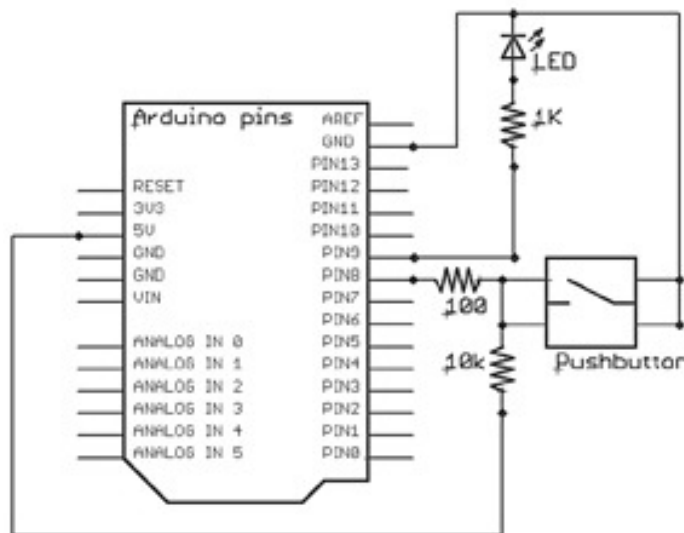
Arduino is, among other things, a programming language. But one of the first things to learn about the Arduino language is that underlying it is the C programming language. What Arduino does is to provide a novice-friendly buffer that hides some standard C requirements and adds some pre-written functions that help the novice use the Arduino board. You don't need to know a thing about C in order to use Arduino or this book. There is an Arduino way of doing things, and that is what we will learn in this section. I just wanted to let you know that most of the stuff we learn here is applicable to C and provides IMHO a good introduction to C if the reader chooses to move up (IMHO) to C.

We will write a simple program and then dissect it to study some of the rules.

## *An Arduino Light Switch*

### **What do we want our program to do**

In the Arduino Quick Start section we used the Arduino Blink example to get an LED blinking on the Arduino board. Some of what we did might even make a bit of sense without having the details explained. Now we are going to add to that program and read a pushbutton switch, and use that reading to turn an LED on or off. Yup, we are going to make an overly complicated and way too expensive light switch. Just remember, our goal at this point is to learn about software, not to create hardware devices that make a lot of sense.



**Figure 17: Schematic – Pushbutton switch and LED**

In hardware we could directly turn the LED on or off with the switch, but since it is a momentary pushbutton switch, it only conducts electricity when pressed. As soon as you release it, the light will go off. But we want to have the button turn the LED on if it is off and off if it is on each time the button is pushed. To do this the micro will need to remember the LED state (is it on or off) and toggle that state each time the button is pushed. (Toggle means that if something is on, turn it off, and if it is off, turn it on.) And since we've got our hardware connected to a PC, we might as well add a couple of features, one to allow us to keep count of the number of times the button has been pressed and another that allows us to turn

the LED on or off from the PC keyboard. These extras will provide extra examples on how to apply the programming rules.

We want this program to:

1. Keep the LED status in memory in a variable named onOff that can be set to either true for on or false for off.
2. Read the status (on or off) of a pushbutton switch.
3. Toggle the value of the onOff variable each time the button is pushed.
4. Turn an LED on or off depending on the value in an onOff variable.

## Set up the hardware

Refer to Figure 17: Schematic – Pushbutton switch and LED, and Figure 18: Drawing - pushbutton switch and LED, and add the parts to the breadboard.

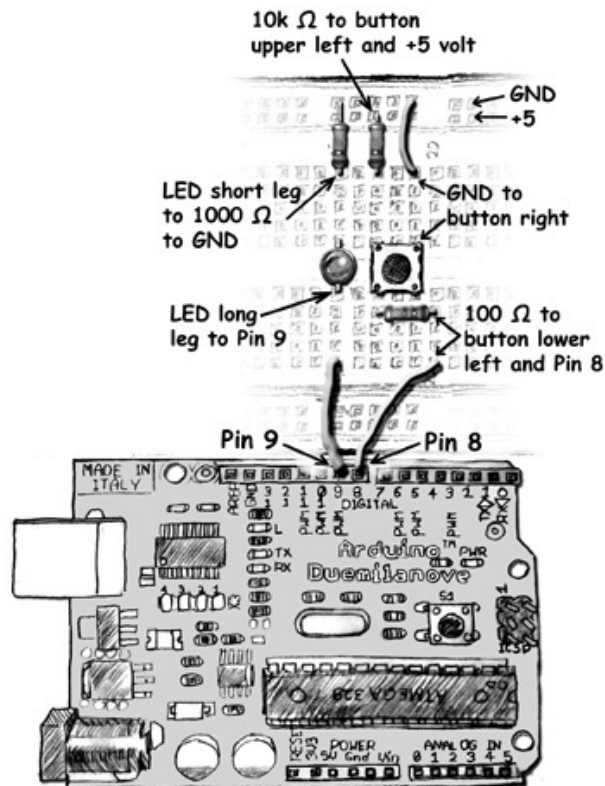


Figure 18: Drawing - pushbutton switch and LED

## Chapter 3: Playing with software – Part 1

Now type the following program into the Arduino IDE.

### **Light\_Switch - source code:**

```
/* Light_Switch Joe Pardue 9/16/09
 *
 * This program was written for novices to demonstrate some
 * of the rules useful for programming with Arduino. It is
 * Instructional code and is not intended for any other purpose.
 */

//
//define constants
//
// pin for LED
#define LEDPIN 9
// pin for pushbutton
#define BUTTONPIN 8
// maximum count to assure the button was really pressed
#define DEBOUNCE 10

//
// define and initialize variables
//
// read button status
int reading = 0;
// count button detected LOW
int count = 0;
// track if LED is on (true) or off (false)
boolean ledState = false;

void setup()
{
  // set button pin to input
  pinMode(BUTTONPIN, INPUT);
  // set LED pin to output
  pinMode(LEDPIN, OUTPUT);
}

void loop()
{
  // wait for a valid button press
  while(count < DEBOUNCE)
  {
    checkButton();
  }
}
```

### Chapter 3: Playing with software – Part 1

```
// reset count
count = 0;

// toggle LED state
if(ledState == true)
{
    // if it was on (true) turn it off (false)
    digitalWrite(LEDPIN, LOW);
    ledState = false;
}
else
{
    // if it was off (false) turn it on (true)
    digitalWrite(LEDPIN, HIGH);
    ledState = true;
}

// give us time to get our finger off the button
delay(500);
}

// Check the button and count up if it is connected to
// ground (LOW) or count down if it is connected to +5 (HIGH)
void checkButton()
{
    // read the pin HIGH or LOW?
    reading = digitalRead(BUTTONPIN);

    // if it is LOW and the count is less than DEBOUNCE
    // increase the count by 1.
    if ( (reading == LOW) && (count < DEBOUNCE) )
    {
        count++;
    }
    else// otherwise if it is greater than 0, decrease by one
    {
        if(count > 0) count--;
    }
}
}
```

If you are really new at this you likely noticed some things in that program that haven't been explained yet, well don't panic they will be explained eventually and

if you wait to write your first program until you understand everything, you'll never write that first program.

I strongly recommend that you type each and every word of this program into the Arduino IDE. While you can get the source code from my website and copy and paste it, you will miss a lot of very important education, not just in getting to carefully look at each character as you type it in, but you will miss out on the debugging process that will result from inevitable typos. Remember debugging is not an optional chore, it is necessary and if you are the type of person who ultimately succeeds, debugging is kind of fun.

### *Some of the rules*

This section takes a very brief look at the Light\_Switch program to help you understand what each line means.

#### **Comments**

You can add comments (text the compiler ignores) to your code two ways. For a single line of comments use double back slashes as in:

```
// give us time to get our finger off the button
delay(500);
```

For multi-line comments, you can begin them with `/*` and end them with `*/` as in:

```
/*
  Light_Switch Joe Pardue 9/16/09

  This program was written for novices to demonstrate some
  of the rules useful for programming with Arduino. It is
  Instructional code and is not intended for any other purpose.
*/
```

#### **Functions**

A function encapsulates a computation. Think of them as building material for programs. A house might be built of studs, nails, and panels. The architect is assured that all 2x4 studs are the same, as are each of the nails and each of the panels, so there is no need to worry about how to make a 2x4 or a nail or a panel,

you just stick them where needed and don't worry how they were made. In the `Light_Switch` program, the `setup()` function twice uses the `pinMode()` function:

```
// set button pin to input
pinMode(BUTTONPIN, INPUT);
```

The writer of the `setup()` function doesn't need to know **how** the `pinMode` function does its job, he only needs to know **what** it does and what parameters to use. In this case the first parameter is the Arduino pin name: `BUTTONPIN` and the second is the mode for using that pin: `INPUT`. Later, you may want to learn a bit more about the function `pinMode()` and you can find out how to use it in the Arduino library reference. Even later you may want to learn how it works and you can find it in the library source code. But the critical point is that **NOW** you can use this function and you don't need to know how it works. It is like the above example of a wood 2x4 used in construction; you don't need to understand how a tree makes wood to build a house.

Encapsulation of code in functions is a key idea in programming and helps make chunks of code more convenient to use. And just as important, it provides a way to make tested code reusable without having to rewrite it for each use.

You are required to use the `setup()` and `loop()` functions in an Arduino program. You are allowed to use the prewritten functions such as `pinMode()` discussed in the Language Reference. And you can build your own functions like the `checkButton` function in the `Light_Switch` program.

When you build your own function such as `checkButton()`, the source code begins with a line that has the data type that the function returns when it is called by another function (the data type is `void` if it doesn't return anything). The data type is followed by the function name, then a parameter list enclosed by parenthesis. (The `checkButton` example doesn't return anything (so it is the `void` type) and doesn't require any parameters) The `pinMode` function required two parameters. The parameter list consists of variables that are provided to the function and used in it. The parameter list is followed by an open curly bracket: '{' and a closed curly bracket: '}' between which you will find the things that the function does.

## Expressions, Statements, and Blocks

**Expressions** are combinations of variables, operators, and function calls that produce a single value. For example:

```
reading = digitalRead(BUTTONPIN)
```

This is an expression that sets the ‘reading’ variable to the value returned by the digitalRead(BUTTONPIN) function.

**Statements** control the program flow and consist of keywords, expressions, and other statements. A semicolon ends a statement. So for example we add a semicolon to our expression and we get a statement:

```
reading = digitalRead(BUTTONPIN);
```

**Blocks** are compound statements grouped by open and closed curly brackets: { }. For example:

```
if(ledState == true)
{
    // if it was on (true) turn it off (false)
    digitalWrite(LEDPIN, LOW);
    ledState = false;
}
```

This groups the two inner statements to be run depending on the condition of the ‘if’ statement. Brackets are also error prone and they can be hard to follow in complex code. Just get used to the process of matching brackets as part of your debugging routine.

## Flow Control

Flow control statements dictate the order in which a series of actions are performed. For example in the loop() function we use ‘while’ to keep us in an inner loop until the count variable (which is incremented in the checkButton() function) becomes equal to (or greater than) the DEBOUNCE constant:

## Chapter 3: Playing with software – Part 1

```
// wait for a valid button press
while(count < DEBOUNCE)
{
    checkButton();
}
```

After running the `checkButton()` function, we exit the while loop and then set the count back to zero . Next we use the ‘if...else’ to decide to turn the LED on or off:

```
// reset count
count = 0;

// toggle LED state
if(ledState == true)
{
    // if it was on (true) turn it off (false)
    digitalWrite(LEDPIN, LOW);
    ledState = false;
}
else
{
    // if it was off (false) turn it on (true)
    digitalWrite(LEDPIN, HIGH);
    ledState = true;
}
```

If you aren’t 100% clear on all this yet, don’t panic, we’ll revisit all these ideas as we move along. You may be especially confused by some of the symbols such as ‘>’ or ‘==’ and well you should be. These things are called operators and will be discussed in more detail in Chapter 7: Playing with software – Part 2.

### **Arduino Functions**

In the Arduino Reference you will see a list of functions that allow you to use the Arduino board resources such as reading and writing to digital and analog pins. We will see how to use many of these functions in the projects section where they will be applied to playing with the hardware in the Smiley Micros Arduino Projects Kit.

