

# Programming Orangutans and the 3pi Robot from the Arduino Environment

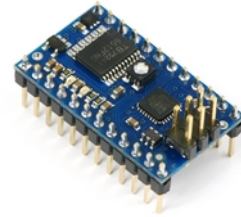


1. Introduction . . . . .	2
2. ATmega168-Arduino Pin Mapping . . . . .	4
2.a. Orangutan SV-168 and LV-168 Pin Mappings . . . . .	4
2.b. Baby Orangutan B Pin Mappings . . . . .	8
2.c. 3pi Robot Pin Mappings . . . . .	11
3. Configuring the Arduino Environment . . . . .	15
4. Using the Arduino IDE: Blinking an LED . . . . .	17
5. Arduino Libraries for the Orangutan and 3pi Robot . . . . .	20
5.a. OrangutanAnalog – Analog Input Library . . . . .	24
5.b. OrangutanBuzzer – High-Level Buzzer Control Library . . . . .	28
5.c. OrangutanLCD – LCD Control Library . . . . .	35
5.d. OrangutanLEDs – LED Control Library . . . . .	40
5.e. OrangutanMotors – Motor Control Library . . . . .	43
5.f. OrangutanPushbuttons – Pushbutton Interface Library . . . . .	47
5.g. Pololu3pi – Sensor Library for the 3pi Robot . . . . .	50

## 1. Introduction



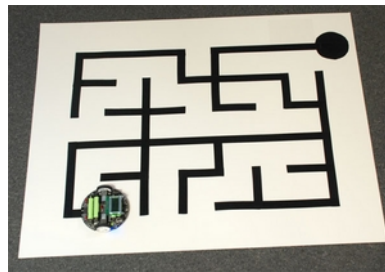
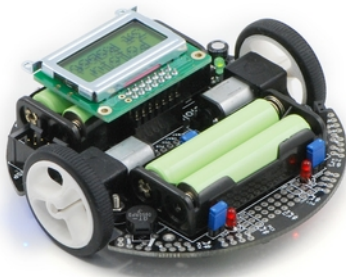
The Orangutan LV-168 attached to a 3-AA battery holder.



Baby Orangutan B-48/168 with included header pins soldered in for breadboard installation.

**Arduino** [<http://www.arduino.cc/>] is a popular, open-source prototyping platform that makes it easy for people with little electronics experience to get into the world of microcontrollers and embedded programming. The Arduino environment consists of an open-source integrated development environment (IDE) and compiler, software libraries, and programmable hardware boards based on the powerful ATmega168 microcontroller. Because our **Orangutan SV-168** [<http://www.pololu.com/catalog/product/1225>], **Orangutan LV-168** [<http://www.pololu.com/catalog/product/775>], and **Baby Orangutan B-168** [<http://www.pololu.com/catalog/product/1216>] robot controllers use this same microcontroller, they have a substantial degree of overlap with Arduinos. Our Orangutan robot controllers can therefore be an attractive alternative to official Arduino hardware for those already familiar with the Arduino environment, and the environment can be a beginner-friendly alternative to AVR Studio for those looking to get started with Orangutans.

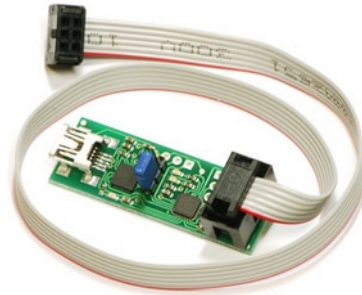
**Note:** the **Pololu 3pi robot** [<http://www.pololu.com/catalog/product/975>] effectively has an Orangutan SV-168 as its core, so this guide and the Orangutan Arduino libraries apply to the 3pi as well. For more information on the 3pi, please see the **3pi user's guide** [<http://www.pololu.com/docs/0J21>].



The Orangutans and 3pi robot offer integrated hardware that Arduinos do not, such as on-board dual motor drivers. This makes them well suited for applications involving control of small- to medium-sized robots. The Orangutan SV-168, Orangutan LV-168, and 3pi robot also have integrated pushbuttons, a piezo buzzer, and a removable LCD, all of which are additional features not found on official Arduinos.

With some minor changes to the Arduino software configuration files, it becomes possible to program our Orangutans and 3pi robot using the Arduino IDE and libraries. This guide will step you through the process of reconfiguring the Arduino IDE, and it will provide a series of custom libraries that will make it easy for you to interface with all of the Orangutan's/ 3pi's onboard hardware.

One key difference between standard Arduino boards and the Orangutan robot controllers is that most Arduinos come with integrated USB-to-serial adapters, which allow you to connect them to your computer and program them through pre-loaded bootloaders, while Orangutans lack integrated USB-to-serial adapters. As such, we recommend you program the Orangutans or 3pi robot using an AVR ISP (referred to as ICSP in the Arduino world) programmer such as our **Orangutan USB programmer** [<http://www.pololu.com/catalog/product/740>]. By taking this approach and leaving off the bootloader, you gain an extra 2 KB of program space (so you have the full 16 KB to work with rather than 14 KB). You also avoid the bootloader delay when you power up or reset the board.



Pololu Orangutan USB Programmer with included cable

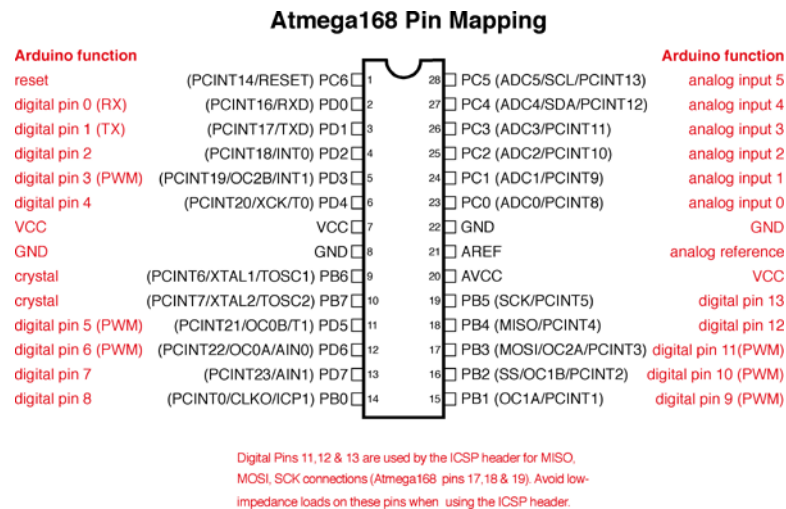
**Note:** This document applies only to the **3pi robot** [<http://www.pololu.com/catalog/product/975>] and **Orangutan SV-168** [<http://www.pololu.com/catalog/product/1225>], **Orangutan LV-168** [<http://www.pololu.com/catalog/product/775>], and **Baby Orangutan B-168** [<http://www.pololu.com/catalog/product/1216>] robot controllers. For simplicity, the latter three devices will be referred to from this point on as “**Orangutans**”, unless a distinction needs to be made between specific Orangutan models. This document does **not** apply to the original **Orangutan** [<http://www.pololu.com/catalog/product/225>] or **Baby Orangutan** [<http://www.pololu.com/catalog/product/215>] robot controllers, nor does it apply to the **Orangutan X2** [<http://www.pololu.com/catalog/product/738>] robot controller.

## 2. ATmega168-Arduino Pin Mapping

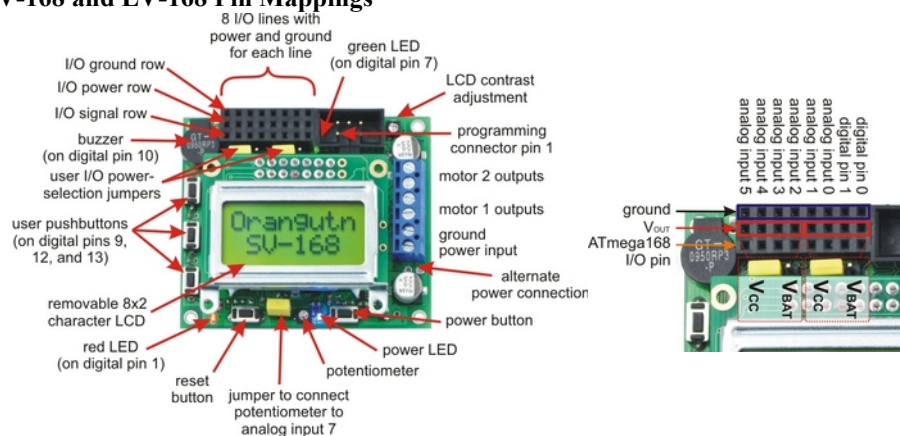
In our standard Orangutan and 3pi documentation, we use the same convention that Atmel uses in its ATmega168 datasheet, referring to pins by their associated ports. For example, pin PD7 is pin 7 of port D. This convention is used because each port is controlled by its own AVR registers, and the bits of these registers correspond to individual pins.

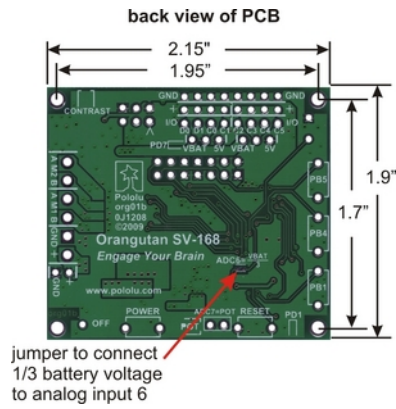
The Arduino insulates the user from this level of detail, referring to the I/O pins as digital pins 0 – 13 and analog inputs 0 – 7. Note that this terminology can be a bit misleading since analog inputs 0 – 5 can also be used as general purpose digital I/O pins (referred to as digital pins 14 – 19).

You can visit <http://www.arduino.cc/en/Hacking/PinMapping168> [<http://www.arduino.cc/en/Hacking/PinMapping168>] to see exactly how the mega168 pins are mapped to Arduino pins. Note that this page shows a 28-pin DIP package chip while the Orangutans carry smaller 32-pin MLF packages that contain two additional dedicated analog inputs: ADC6 and ADC7. In the Arduino environment, these pins become arduino analog inputs 6 and 7, respectively. The following diagram comes from the page linked above:

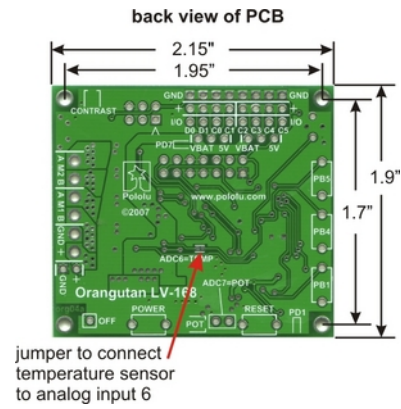


### 2.a. Orangutan SV-168 and LV-168 Pin Mappings





Orangutan SV-168 bottom view with dimensions (Arduino pin mapping).



Orangutan LV-168 bottom view with dimensions (Arduino pin mapping).

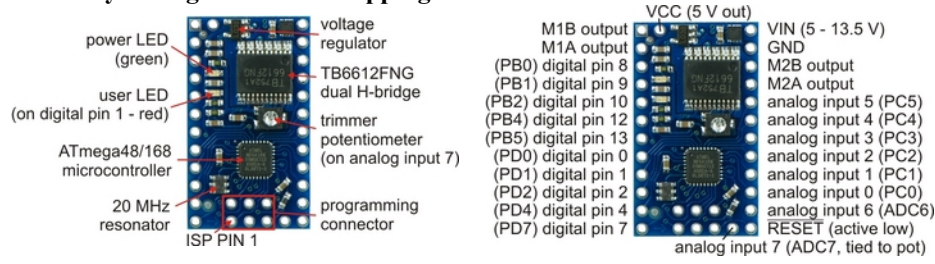
## Pin Assignment Table Sorted by Function

Function	Arduino Pin	mega168 Pin
digital I/Os (x8)	digital pins 0 and 1 analog inputs 0 – 5	PD0, PD1, PC0 – PC5
analog inputs (x8)	analog inputs 0 – 7	PC0 – PC5, ADC6, ADC7
motor 1 control (A and B)	digital pins 5 and 6	PD5 and PD6
motor 2 control (A and B)	digital pins 3 and 11	PD3 and PB3
red user LED	digital pin 1	PD1
green user LED	digital pin 7	PD7
user pushbuttons (x3)	digital inputs 9, 12, and 13	PB1, PB4, and PB5
buzzer	digital pin 10	PB2
LCD control (RS, R/W, E)	digital pins 2, 8, and 4	PD2, PB0, and PD4
LCD data (4-bit: DB4 – DB7)	digital pins 9, 12, 13, and 7	PB1, PB4, PB5, and PD7
user trimmer potentiometer	analog input 7 (through jumper)	ADC7
temperature sensor (LV-168 only) battery voltage monitor (SV-168 only)	analog input 6 (through SMT jumper)	ADC6
ICSP programming lines (x3)	digital pins 11, 12, and 13	PB3, PB4, PB5
reset pushbutton	reset	PC6
UART (RX and TX)	digital pins 0 and 1	PD0 and PD1
I2C/TWI (SDA and SCL)	analog inputs 4 and 5	PC4 and PC5
SPI	inaccessable to user	

## Pin Assignment Table Sorted by Pin

Arduino Pin	Orangutan Function	Notes/Alternate Functions
digital pin 0	digital I/O	USART input pin (RXD)
digital pin 1	digital I/O	<b>connected to red user LED</b> (high turns LED on) USART output pin (TXD)
digital pin 2	<b>LCD control line RS</b>	external interrupt 0 (INT0)
digital pin 3	<b>M2 control line</b>	Timer2 PWM output B (OC2B)
digital pin 4	<b>LCD control line E</b>	USART external clock input/output (XCK) Timer0 external counter (T0)
digital pin 5	<b>M1 control line</b>	Timer0 PWM output B (OC0B)
digital pin 6	<b>M1 control line</b>	Timer0 PWM output A (OC0A)
digital pin 7	<b>LCD data line DB7</b>	<b>connected to green user LED</b> (high turns LED on)
digital pin 8	<b>LCD control line R/W</b>	Timer1 input capture (ICP1) divided system clock output (CLK0)
digital pin 9	<b>LCD data line DB4</b>	<b>user pushbutton</b> (pressing pulls pin low) Timer1 PWM output A (OC1A)
digital pin 10	<b>buzzer</b>	Timer1 PWM output B (OC1B)
digital pin 11	<b>M2 control line</b>	Timer2 PWM output A (OC2A) ISP programming line
digital pin 12	<b>LCD data line DB5</b>	<b>user pushbutton</b> (pressing pulls pin low) <b>Caution: also an ISP programming line</b>
digital pin 13	<b>LCD data line DB6</b>	<b>user pushbutton</b> (pressing pulls pin low) <b>Caution: also an ISP programming line</b>
analog input 0	analog input and digital I/O	ADC input channel 0 (ADC0)
analog input 1	analog input and digital I/O	ADC input channel 1 (ADC1)
analog input 2	analog input and digital I/O	ADC input channel 2 (ADC2)
analog input 3	analog input and digital I/O	ADC input channel 3 (ADC3)
analog input 4	analog input and digital I/O	ADC input channel 4 (ADC4) I2C/TWI input/output data line (SDA)
analog input 5	analog input and digital I/O	ADC input channel 5 (ADC5) I2C/TWI clock line (SCL)
analog input 6	dedicated analog input	<b>SMT-jumpered to temperature sensor</b> (LV-168 only) <b>SMT-jumpered to battery voltage monitor</b> (SV-168 only)ADC input channel 6 (ADC6)
analog input 7	dedicated analog input	<b>jumpered to user trimmer potentiometer</b> ADC input channel 7 (ADC7)
reset	reset pushbutton	internally pulled high; active lowdigital I/O disabled by default

### 2.b. Baby Orangutan B Pin Mappings





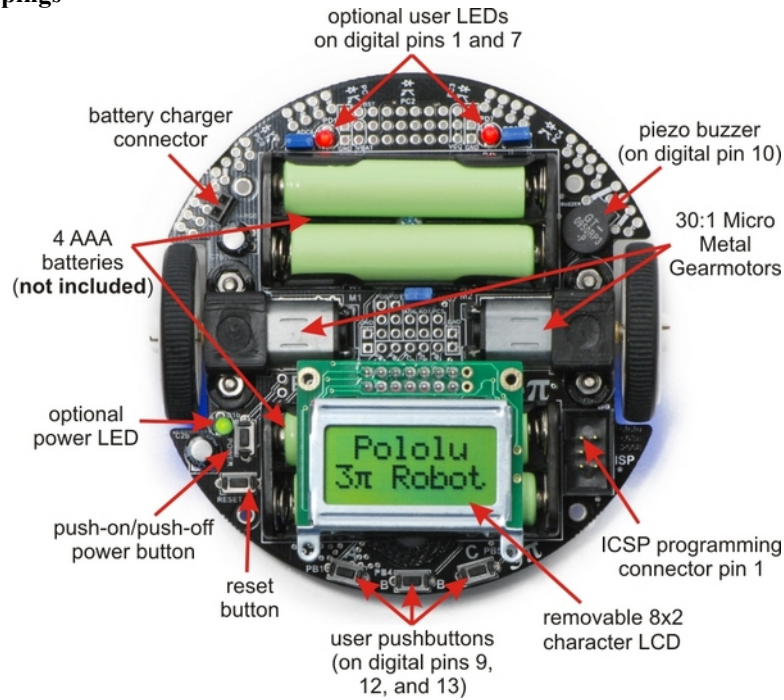
## Pin Assignment Table Sorted by Function

Function	Arduino Pin	mega168 Pin
digital I/Os (x16)	digital pins 0 – 2, 4, 7 – 10, 12, 13 analog inputs 0 – 5	PD0 – PD2, PD4, PD7, PB0 – PB2, PB4, PB5, PC0 – PC5
analog inputs (x8)	analog inputs 0 – 7	PC0 – PC5, ADC6, ADC7
motor 1 control (A and B)	digital pins 5 and 6	PD5 and PD6
motor 2 control (A and B)	digital pins 3 and 11	PD3 and PB3
red user LED	digital pin 1	PD1
user trimmer potentiometer	analog input 7	ADC7
ICSP programming lines (x3)	digital pins 11, 12, and 13	PB3, PB4, PB5
$\overline{\text{RESET}}$	reset	PC6
UART (RX and TX)	digital pins 0 and 1	PD0 and PD1
I2C/TWI (SDA and SCL)	analog inputs 4 and 5	PC4 and PC5
SPI	inaccessable to user	
Timer1 PWM outputs (A and B)	digital pins 9 and 10	PB1 and PB2

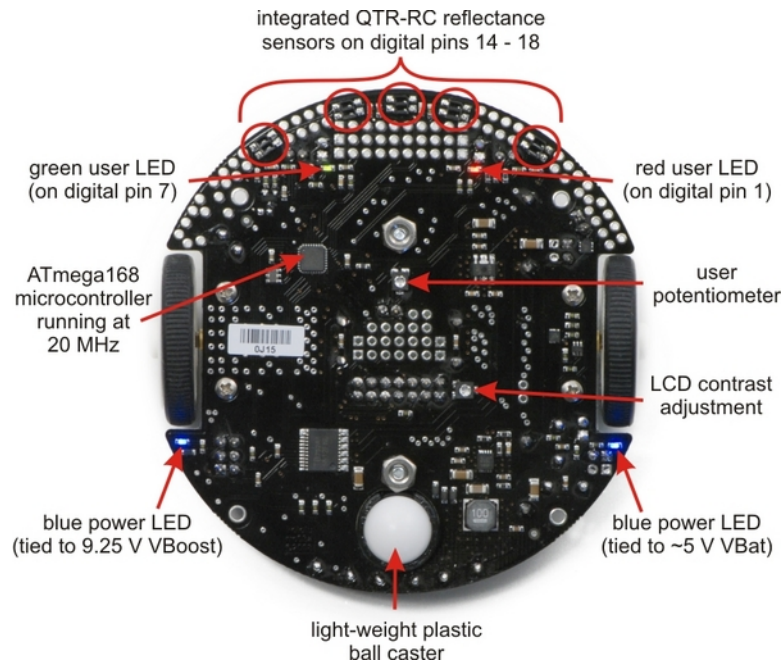
## Pin Assignment Table Sorted by Pin

Arduino Pin	Orangutan Function	Notes/Alternate Functions
digital pin 0	digital I/O	USART input pin (RXD)
digital pin 1	digital I/O	<b>connected to red user LED</b> (high turns LED on) USART output pin (TXD)
digital pin 2	digital I/O	external interrupt 0 (INT0)
digital pin 3	<b>M2 control line</b>	Timer2 PWM output B (OC2B)
digital pin 4	digital I/O	USART external clock input/output (XCK) Timer0 external counter (T0)
digital pin 5	<b>M1 control line</b>	Timer0 PWM output B (OC0B)
digital pin 6	<b>M1 control line</b>	Timer0 PWM output A (OC0A)
digital pin 7	digital I/O	
digital pin 8	digital I/O	Timer1 input capture (ICP1) divided system clock output (CLK0)
digital pin 9	digital I/O	Timer1 PWM output A (OC1A)
digital pin 10	digital I/O	Timer1 PWM output B (OC1B)
digital pin 11	<b>M2 control line</b>	Timer2 PWM output A (OC2A) ISP programming line
digital pin 12	digital I/O	<b>Caution: also an ISP programming line</b>
digital pin 13	digital I/O	<b>Caution: also an ISP programming line</b>
analog input 0	analog input and digital I/O	ADC input channel 0 (ADC0)
analog input 1	analog input and digital I/O	ADC input channel 1 (ADC1)
analog input 2	analog input and digital I/O	ADC input channel 2 (ADC2)
analog input 3	analog input and digital I/O	ADC input channel 3 (ADC3)
analog input 4	analog input and digital I/O	ADC input channel 4 (ADC4) I2C/TWI input/output data line (SDA)
analog input 5	analog input and digital I/O	ADC input channel 5 (ADC5) I2C/TWI clock line (SCL)
analog input 6	dedicated analog input	ADC input channel 6 (ADC6)
analog input 7	dedicated analog input	<b>connected to user trimmer potentiometer</b> ADC input channel 7 (ADC7)
reset	$\overline{\text{RESET}}$ pin	internally pulled high; active low digital I/O disabled by default

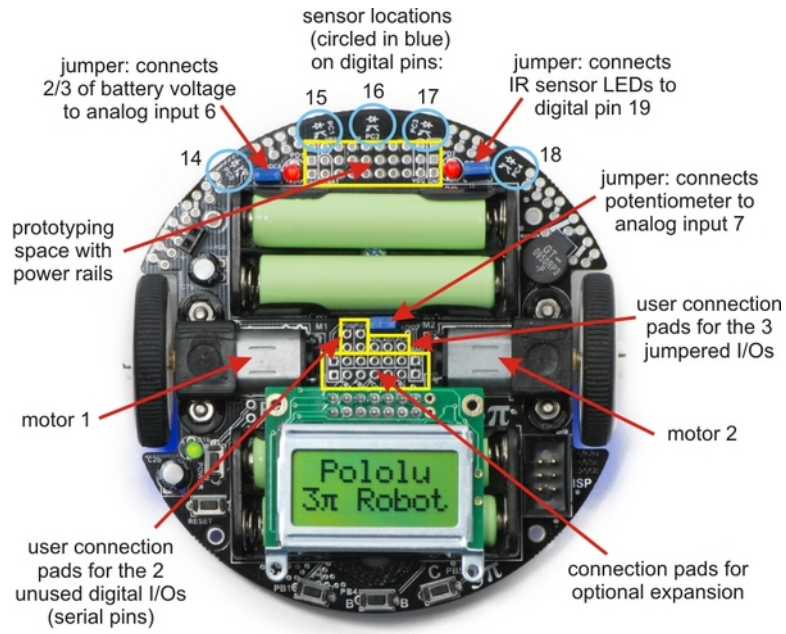
### 2.c. 3pi Robot Pin Mappings



General features of the Pololu 3pi robot, top view.



Labeled bottom view of the Pololu 3pi robot.



Specific features of the Pololu 3pi robot, top view.

## Pin Assignment Table Sorted by Function

Function	Arduino Pin	mega168 Pin
free digital I/Os (x3) (remove PC5 jumper to free digital pin 19)	digital pins 0, 1, 19	PD0, PD1, PC5
free analog inputs (if you remove jumpers, x3)	analog inputs 5 – 7	PC5, ADC6, ADC7
motor 1 (left motor) control (A and B)	digital pins 5 and 6	PD5 and PD6
motor 2 (right motor) control (A and B)	digital pins 3 and 11	PD3 and PB3
QTR-RC reflectance sensors (left to right, x5)	digital pins 14 – 18	PC0 – PC4
red (left) user LED	digital pin 1	PD1
green (right) user LED	digital pin 7	PD7
user pushbuttons (left to right, x3)	digital inputs 9, 12, and 13	PB1, PB4, and PB5
buzzer	digital pin 10	PB2
LCD control (RS, R/W, E)	digital pins 2, 8, and 4	PD2, PB0, and PD4
LCD data (4-bit: DB4 – DB7)	digital pins 9, 12, 13, and 7	PB1, PB4, PB5, and PD7
reflectance sensor IR LED control (drive low to turn IR LEDs off)	digital pin 19 (through jumper)	PC5
user trimmer potentiometer	analog input 7 (through jumper)	ADC7
2/3rds of battery voltage	analog input 6 (through jumper)	ADC6
ICSP programming lines (x3)	digital pins 11, 12, and 13	PB3, PB4, PB5
reset pushbutton	reset	PC6
UART (RX and TX)	digital pins 0 and 1	PD0 and PD1
I2C/TWI	inaccessible to user	
SPI	inaccessible to user	

## Pin Assignment Table Sorted by Pin

Arduino Pin	Orangutan Function	Notes/Alternate Functions
digital pin 0	free digital I/O	USART input pin (RXD)
digital pin 1	free digital I/O	<b>connected to red user LED</b> (high turns LED on) USART output pin (TXD)
digital pin 2	<b>LCD control line RS</b>	external interrupt 0 (INT0)
digital pin 3	<b>M2 control line</b>	Timer2 PWM output B (OC2B)
digital pin 4	<b>LCD control line E</b>	USART external clock input/output (XCK) Timer0 external counter (T0)
digital pin 5	<b>M1 control line</b>	Timer0 PWM output B (OC0B)
digital pin 6	<b>M1 control line</b>	Timer0 PWM output A (OC0A)
digital pin 7	<b>LCD data line DB7</b>	<b>connected to green user LED</b> (high turns LED on)
digital pin 8	<b>LCD control line R/W</b>	Timer1 input capture (ICP1) divided system clock output (CLK0)
digital pin 9	<b>LCD data line DB4</b>	<b>user pushbutton</b> (pressing pulls pin low) Timer1 PWM output A (OC1A)
digital pin 10	<b>buzzer</b>	Timer1 PWM output B (OC1B)
digital pin 11	<b>M2 control line</b>	Timer2 PWM output A (OC2A) ISP programming line
digital pin 12	<b>LCD data line DB5</b>	<b>user pushbutton</b> (pressing pulls pin low) <b>Caution: also an ISP programming line</b>
digital pin 13	<b>LCD data line DB6</b>	<b>user pushbutton</b> (pressing pulls pin low) <b>Caution: also an ISP programming line</b>
digital pin 14	<b>QTR-RC reflectance sensor</b>	(drive high for 10 us, then wait for line input to go low) sensor labeled PC0 (leftmost sensor)
digital pin 15	<b>QTR-RC reflectance sensor</b>	(drive high for 10 us, then wait for line input to go low) sensor labeled PC1
digital pin 16	<b>QTR-RC reflectance sensor</b>	(drive high for 10 us, then wait for line input to go low) sensor labeled PC2 (center sensor)
digital pin 17	<b>QTR-RC reflectance sensor</b>	(drive high for 10 us, then wait for line input to go low) sensor labeled PC3
digital pin 18	<b>QTR-RC reflectance sensor</b>	(drive high for 10 us, then wait for line input to go low) sensor labeled PC4 (rightmost sensor)
digital pin 19	analog input and digital I/O	<b>jumpered to sensors' IR LEDs</b> (driving low turns off emitters) ADC input channel 5 (ADC5)
analog input 6	dedicated analog input	<b>jumpered to 2/3rds of battery voltage</b> ADC input channel 6 (ADC6)
analog input 7	dedicated analog input	<b>jumpered to user trimmer potentiometer</b> ADC input channel 7 (ADC7)
reset	reset pushbutton	internally pulled high; active low digital I/O disabled by default

### 3. Configuring the Arduino Environment

**Note:** The configuration instructions in this section explain how to alter the Arduino IDE so that it programs your controller using an ICSP rather than an Arduino bootloader. If you would rather use the bootloader, you should skip steps 2 – 4. You will still need to use an ICSP to initially load the bootloader onto your Orangutan or 3pi, and you will also need to obtain a USB-to-serial adapter or RS-232-to-TTL serial adapter through which the Arduino software can talk to the Arduino bootloader on your Orangutan. The Orangutan USB programmer can serve as such a USB-to-serial adapter if you switch the mode jumper. Be aware that the Baby Orangutan B-48 has an ATmega48 microcontroller, which lacks a bootloader section, meaning the B-48 version can only be programmed using an ICSP.

1. Go to <http://www.arduino.cc/en/Main/Software> [<http://www.arduino.cc/en/Main/Software>] and download the latest Arduino software for your platform (Arduino 0012 as of this writing). Extract the files and run the Arduino IDE once to generate a preferences file (this will happen automatically when the program is first run). Close the Arduino IDE.
2. Add the following to the end of the **boards.txt** file in the **arduino-0012/hardware** directory:

```
orangutan.name=Orangutan with Programmer
orangutan.upload.maximum_size=16384
orangutan.build.mcu=atmega168
orangutan.build.f_cpu=20000000L
orangutan.build.core=arduino
```

or replace your **boards.txt** file with [this](http://www.pololu.com/file/download/boards.txt?file_id=0J93) [[http://www.pololu.com/file/download/boards.txt?file\\_id=0J93](http://www.pololu.com/file/download/boards.txt?file_id=0J93)] (3k txt).

This will add an **Orangutan with Programmer** option to the **Tools > Board** menu of the Arduino IDE. It accounts for the 20 MHz clock speed (standard Arduino boards run at 8 or 16 MHz) and bigger flash size (due to lack of an Arduino bootloader) on the Orangutans and 3pi.

**Note:** If you are using a Baby Orangutan B-48, you need to modify this new file entry slightly. The maximum upload size should be **4096** rather than 16384 and the build mcu should be **atmega48** rather than atmega168.

3. If you are using our Orangutan USB programmer, add the following to the end of the **programmers.txt** file in the **arduino-0012/hardware** directory:

```
avrispv2.name=AVR ISP v2
avrispv2.communication=serial
avrispv2.protocol=avrispv2
```

or replace your **programmers.txt** file with [this](http://www.pololu.com/file/download/programmers.txt?file_id=0J94) [[http://www.pololu.com/file/download/programmers.txt?file\\_id=0J94](http://www.pololu.com/file/download/programmers.txt?file_id=0J94)] (0k txt).

This will allow the Arduino IDE to program with the avrispv2 protocol used by our Orangutan USB programmer. If you have a different ICSP programmer, such as AVR ISP mkII, check to see if its protocol already exists in the **programmers.txt** file. If it doesn't, add your programmer to the to the file by creating a new entry for it. For

example:

```
mynewprogrammer.name=Whatever I want to call it
mynewprogrammer.communication=serial
mynewprogrammer.protocol=<insert protocol here>
```

4. Edit your Arduino IDE's **preferences.txt** file. You can find this file in the following location, depending on your platform:
  - **Windows:** c:\Documents and Settings\<<USERNAME>\Application Data\Arduino\preferences.txt
  - **Mac:** /Users//Library/Arduino/preferences.txt
  - **Linux:** ~/.arduino/preferences.txt

Note that the **Application Data** directory on the Windows platform is typically hidden, so you might need to edit your folder properties so that hidden files are displayed.

Open this file and replace the line **upload.using=bootloader** with:

- **upload.using=avrispv2** if you are using the Orangutan USB programmer
- **upload.using=avrispmkii** if you are using the an AVR ISP mkII
- **upload.using=mynewprogrammer** if you are using something else you added to programmers.txt



## 4. Using the Arduino IDE: Blinking an LED

Now it's time to take the Arduino IDE for a spin. Our first goal will be to load a simple LED-blinking sketch (Arduino program) onto our Orangutan or 3pi.

**Open the Arduino IDE and create the following sketch:**

```

/*
 * Blink
 *
 * The basic Arduino example, modified to work for Orangutans.
 * Turns on an LED for one second, then off for one second,
 * and so on... We use pin 1 because Orangutans have an LED
 * on PD1, which corresponds to Arduino pin 1.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */

int ledPin = 1;           // LED connected to digital pin 1 (PD1) on Orangutans

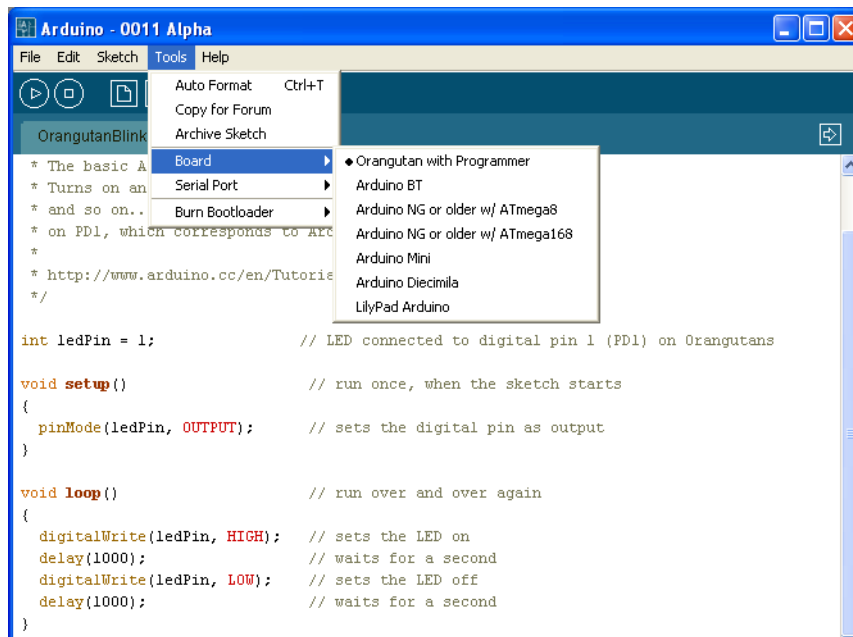
void setup()              // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}

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
}

```

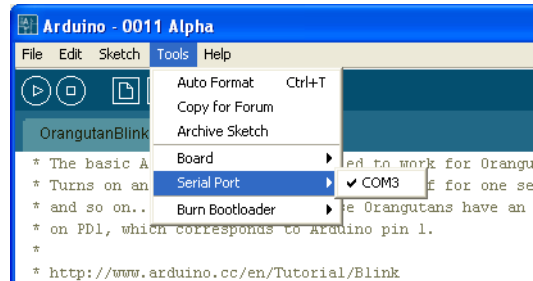
Either copy the above text into a new, blank sketch, or download this sketch as **OrangutanBlinkExample.pde** [[http://www.pololu.com/file/download/OrangutanBlinkExample.pde?file\\_id=0J95](http://www.pololu.com/file/download/OrangutanBlinkExample.pde?file_id=0J95)] (1k pde) and open it in the Arduino IDE.

**Select our specially created Arduino board using the Tools->Board menu:**

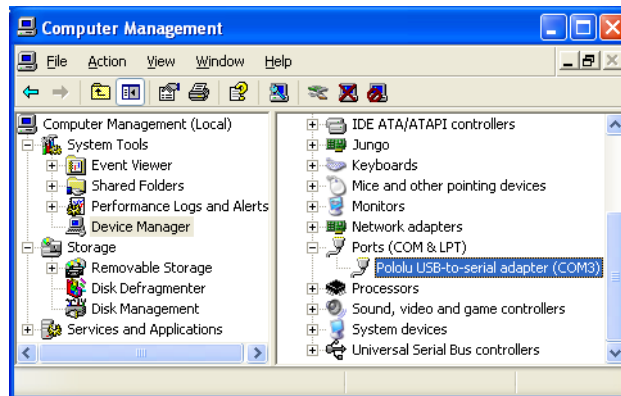


The board we created by modifying the **boards.txt** file is called **Orangutan with Programmer**. This is the setting we want to use for programming an Orangutan or 3pi robot.

### Select the programmer's serial port:



If you are using an Orangutan USB programmer, make sure you have installed the proper drivers (see the **programmer's user's guide** [<http://www.pololu.com/docs/0J6>]) and have connected it to your computer. Once this is done, you should be able to see its COM port listed under the **Tools > Serial Port** menu. If you don't know which COM port your programmer is on, you can determine this by bringing up your computer's device manager and expanding the **Ports (COM & LPT)** list. To bring up the device manager, right-click on "My Computer" and select **Manage**, then click the **Device Manager** option listed on the left side under **System Tools**.

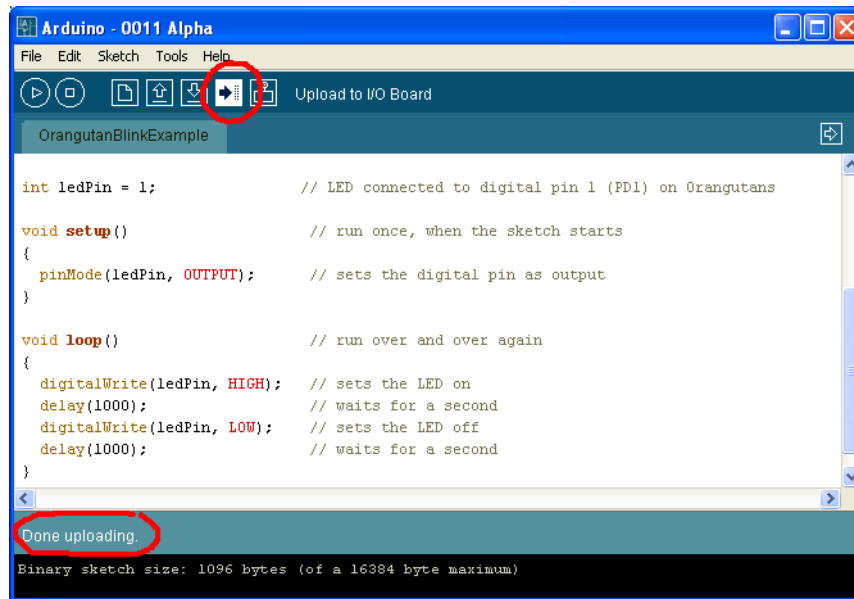


If you are using a true USB programmer (i.e. one that does not emulate a serial port), you shouldn't need to select a serial port.

### Program your Orangutan or 3pi with the sketch:

Connect your programmer to the Orangutan or 3pi via its six-pin ISP header, making sure that pin 1 of the programmer lines up with pin 1 of the target's programming header. The Orangutan SV-168, Orangutan LV-168, and 3pi each have a shrouded header that will enforce correct pin orientation, but the Baby Orangutan B does not. Pin 1 on the Baby Orangutan is marked by a copper arrow on the PCB and pin 1 on our Orangutan USB programmer is marked by a raised plastic triangle on the IDC connector.

With the programmer connected to both your computer and your Orangutan or 3pi, click the **Upload to I/O Board** button (circled in the screen capture below) or press ctrl-U to compile the sketch and upload it to your target device. The status bar at the bottom of the window will say "Uploading to I/O Board..." while the upload is in progress and will read "Done uploading." when the upload is complete. If everything has worked correctly you should now see the target's red user LED blinking on and off every second. Please note that if you are using a 3pi and have not yet soldered in the optional through-hole LEDs, you will need to look at the underside of the robot to see the flashing LED.



## 5. Arduino Libraries for the Orangutan and 3pi Robot

**Note:** Because the Orangutans were not designed as official Arduino boards, not all existing Arduino library code will be directly compatible with them. For example, certain Arduino functions like **delayMicroseconds()** and **pulseIn()** are designed to run at 16 MHz and hence their behavior is altered by the Orangutan's 20 MHz clock. In other cases, Orangutan-specific libraries might conflict with existing Arduino libraries if the two are competing for the same mega168 hardware peripherals (e.g. two libraries that try to use Timer1 in different ways will not be compatible).

## Overview

This section provides a series of Orangutan-specific libraries that will allow you to easily interface with the Orangutan/3pi hardware from the Arduino environment. It is important to note that the Baby Orangutan B is a hardware subset of the larger Orangutan SV-168 and LV-168, and these Orangutans are a hardware subset of the 3pi robot. What this means is that any library code that works on the Baby Orangutan B will also work on the Orangutan and 3pi robot, and any library code that works on the Orangutan will also work on the 3pi. The following subsections contain detailed documentation of each Orangutan Arduino library. **All of the libraries that apply to the Orangutan will work for the corresponding hardware on the 3pi robot.**

The only library class that can be instantiated multiple times is PololuQTRSensors. You should never be using more than one instance of any other library object, so to that end all of the remaining Orangutan/3pi Arduino libraries have made their objects static. This means that there are two possible ways for you to call these library's methods:

1. OrangutanLibrary library;  
library.method();
2. OrangutanLibrary::method();

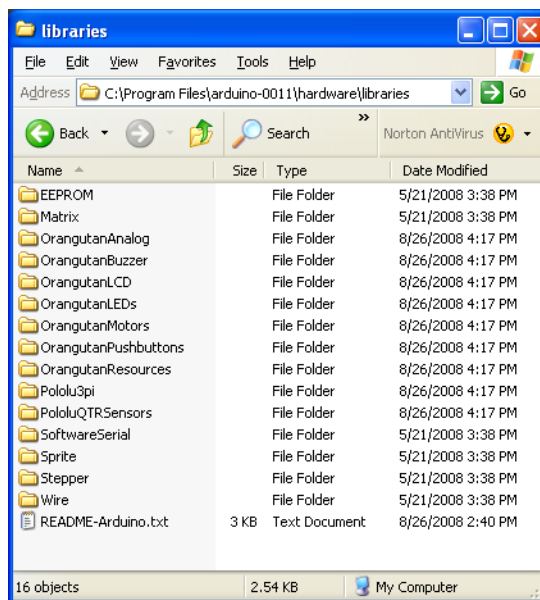
The only library classes that need to be explicitly initialized before they are used are PololuQTRSensors and Pololu3pi, since these objects are initialized with user-specified parameters. None of the remaining Orangutan library objects needs to be explicitly initialized in your **setup()** function as initialization is automatically performed when needed.

## Installation

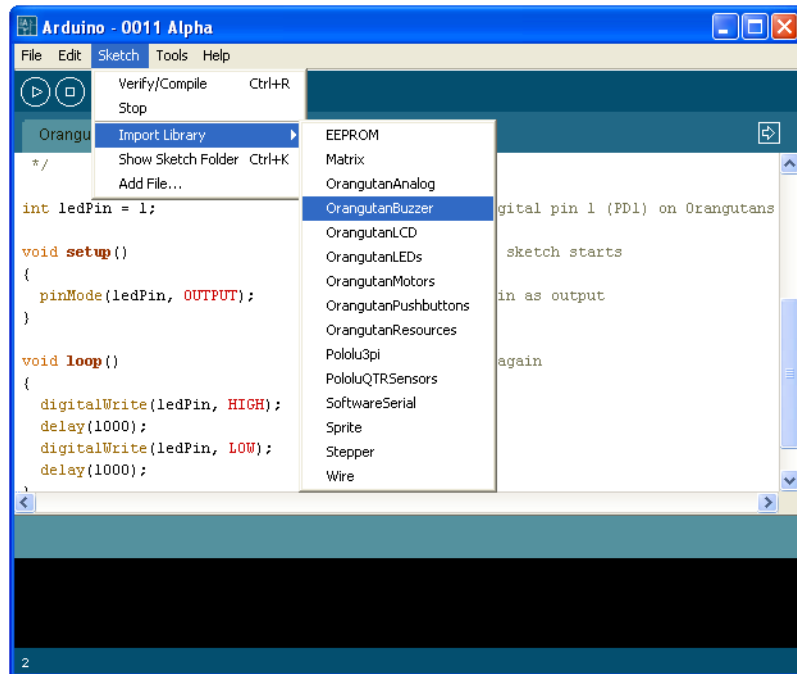
**Note:** If you currently have an older version of our Pololu Arduino Libraries, your first step should be to delete all of the Orangutan\_\_ and Pololu\_\_ directories from your **arduino-0012/hardware/libraries** directory. If you don't perform this step, the newer version of the libraries might not get compiled. Additionally, the **080929** [[http://www.pololu.com/file/download/PololuArduinoLibraries-080929.zip?file\\_id=0J134](http://www.pololu.com/file/download/PololuArduinoLibraries-080929.zip?file_id=0J134)] (86k zip) version of the Pololu Arduino libraries will only work correctly with Arduino version 0012 or later. If you have an older version of the Arduino environment, you either need to update to version 0012 or you can download version **080826** [[http://www.pololu.com/file/download/PololuArduinoLibraries-080826.zip?file\\_id=0J127](http://www.pololu.com/file/download/PololuArduinoLibraries-080826.zip?file_id=0J127)] (80k zip) of the Pololu Arduino libraries, which works with Arduino version 0011 but not 0012

Download the following archive: **PololuArduinoLibraries-080929.zip** [[http://www.pololu.com/file/download/PololuArduinoLibraries-080929.zip?file\\_id=0J134](http://www.pololu.com/file/download/PololuArduinoLibraries-080929.zip?file_id=0J134)] (86k zip)

and extract it to your **arduino-0012/hardware/libraries** directory. Once this is complete you should see a series of new directories have appeared in the extraction directory. The screen shot below shows where the library directories should appear once they are extracted:

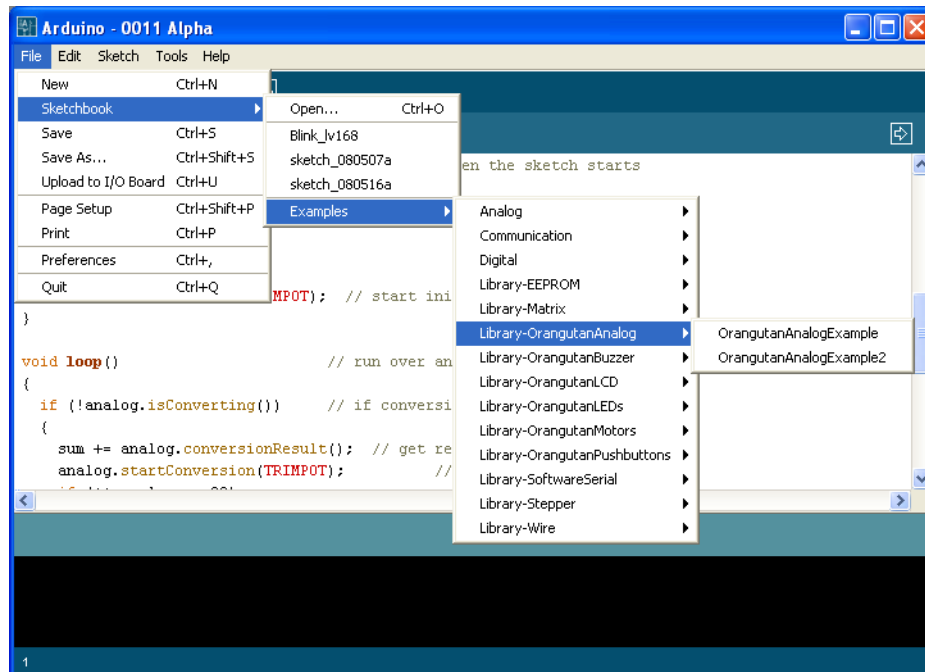


You should now be able to use these libraries in your sketches. For example, if you want to use the OrangutanBuzzer library, you can select **Sketch > Import Library > OrangutanBuzzer** from your Arduino 0012 IDE, or you can simply type **#include <OrangutanBuzzer.h>** at the top of your sketch. Note that you might need to restart your Arduino 0012 IDE before it sees the new libraries. If you do not need the functionality of a specific library, you should leave it out of your sketch (simply delete the **#include** line associated with it from the top of your sketch if you've already added it). Included libraries will take up program space, even if you don't use them in your program.



## Usage Examples

Most libraries come with at least one example sketch that demonstrates how the library can be used. To load an example sketch, open the Arduino IDE and go to **File > Sketchbook > Examples > Library-Orangutan\_\_\_\_\_**.



The Pololu3pi library comes with several detailed line-following and maze-solving example sketches that can be loaded using **File > Sketchbook > Examples > Library-Pololu3pi**.

### 5.a. OrangutanAnalog - Analog Input Library



## Overview

This library provides a set of methods that can be used to read analog voltage inputs, as well as functions specifically designed to read the value of the trimmer potentiometer (on the **3pi robot** [<http://www.pololu.com/catalog/product/975>], **Orangutan SV-168** [<http://www.pololu.com/catalog/product/1225>], **Orangutan LV-168** [<http://www.pololu.com/catalog/product/775>], and **Baby Orangutan B** [<http://www.pololu.com/catalog/product/1216>]), the battery voltage level in millivolts (on the 3pi only), and the value of the temperature sensor in tenths of a degree F or C (on the Orangutan LV-168 only). This library gives you more control than existing Arduino analog input functions.

You do not need to initialize your OrangutanAnalog object before use. All initialization is performed automatically when needed.

All of the methods in this class are static; you should never have more than one instance of an OrangutanAnalog object in your sketch.

## OrangutanAnalog Methods

Complete documentation of this library's methods can be found in **Section 3** of the **Pololu AVR Library Command Reference** [<http://www.pololu.com/docs/0J18>].

## Usage Examples

This library comes with two example sketches that you can load by going to **File > Sketchbook > Examples > Library-OrangutanAnalog**. The example sketches that come with the **OrangutanMotors** library also make limited use of this library.

### 1. OrangutanAnalogExample

Demonstrates how you can use the methods in this library to read the analog voltage of the trimmer potentiometer in the background while the rest of your code executes. If the ADC is free, the program starts a conversion on the **TRIMPOT** analog input (channel 7), and then it proceeds to execute the rest of the code in `loop()` while the ADC hardware works. Polling of the `isConverting()` method allows the program to determine when the conversion is complete and to update its notion of the trimpot value accordingly. Feedback is given via the red user LED, whose brightness is made to scale with the trimpot position. This example will work for the 3pi robot, Orangutan SV-168, Orangutan LV-168, and Baby Orangutan B.

```
#include <OrangutanLEDs.h>
#include <OrangutanAnalog.h>

/*
 * OrangutanAnalogExample
 *
 * This sketch uses the OrangutanAnalog library to read the voltage output
 * of the trimpot in the background while the rest of the main loop executes.
 * The LED is flashed so that its brightness appears proportional to the
 * trimpot position. This example will work on the 3pi robot, Orangutan SV-168,
 * Orangutan LV-168, and Baby Orangutan B.
 *
 * http://www.pololu.com/docs/0J17/5.a
 * http://www.pololu.com
 * http://forum.pololu.com
 */

OrangutanLEDs leds;
OrangutanAnalog analog;

unsigned int sum;
unsigned int avg;
unsigned char samples;

void setup()                // run once, when the sketch starts
{
  analog.setMode(MODE_8_BIT); // 8-bit analog-to-digital conversions
  sum = 0;
  samples = 0;
  avg = 0;
  analog.startConversion(TRIMPOT); // start initial conversion
}

void loop()                 // run over and over again
{
  if (!analog.isConverting()) // if conversion is done...
  {
    sum += analog.conversionResult(); // get result
    analog.startConversion(TRIMPOT); // and start next conversion
    if (++samples == 20)
    {
      avg = sum / 20; // compute 20-sample average of ADC result
      samples = 0;
      sum = 0;
    }
  }

  // when avg == 0, the red LED is almost totally off
  // when avg == 255, the red LED is almost totally on
  // brightness should scale approximately linearly in between
  leds.red(LOW); // red LED off
  delayMicroseconds(256 - avg);
  leds.red(HIGH); // red LED on
}
```

```
    delayMicroseconds(avg + 1);
}
```

## 2. OrangutanAnalogExample2

Intended for use on the Orangutan LV-168. Note that it will run on the 3pi robot and Orangutan SV-168, but the displayed temperature will be incorrect as the analog input connected to the temperature sensor on the Orangutan LV-168 is connected to 2/3rds of the battery voltage on the 3pi and to 1/3rd of the battery voltage on the Orangutan SV-168. It displays on the LCD the trimmer potentiometer output in millivolts and the temperature sensor output in degrees Fahrenheit. If you hold a finger on the underside of the Orangutan LV-168's PCB near the center of the board, you should see the temperature reading slowly start to rise. Be careful not to zap the board with electrostatic discharge if you try this!

```
#include <OrangutanLCD.h>
#include <OrangutanAnalog.h>

/*
 * OrangutanAnalogExample2
 *
 * This sketch uses the OrangutanAnalog library to read the voltage output
 * of the trimpot (in millivolts) and to read the Orangutan LV-168's
 * temperature sensor in degrees Fahrenheit. These values are printed to
 * the LCD 10 times per second. This example is intended for use with the
 * Orangutan LV-168, though all but the temperature-measuring portion
 * will work on the 3pi robot (on the 3pi, analog input 6 connects to 2/3rds
 * of the battery voltage rather than a temperature sensor) and the
 * Orangutan SV-168 (on the SV-168, analog input 6 connects to 1/3rd of
 * the battery voltage).
 *
 * You should see the trimpot voltage change as you turn it, and you can
 * get the temperature reading to slowly increase by holding a finger on the
 * underside of the Orangutan LV-168's PCB near the center of the board.
 * Be careful not to zap the board with electrostatic discharge if you
 * try this!
 */

OrangutanLCD lcd;
OrangutanAnalog analog;

void setup()                // run once, when the sketch starts
{
  analog.setMode(MODE_10_BIT); // 10-bit analog-to-digital conversions
}

void loop()                 // run over and over again
{
  lcd.gotoXY(0,0);         // LCD cursor to home position (upper-left)
  lcd.print(analog.toMillivolts(analog.readTrimpot())); // trimpot output in mV
  lcd.print(" mV ");      // added spaces are to overwrite left over chars

  lcd.gotoXY(0, 1);       // LCD cursor to start of the second line

  // get temperature in tenths of a degree F
  unsigned int temp = analog.readTemperatureF();
  lcd.print(temp/10);      // get the whole number of degrees
  lcd.print('.');         // print the decimal point
  lcd.print(temp - (temp/10)*10); // print the tenths digit
  lcd.print((char)223);   // print a degree symbol
  lcd.print("F ");       // added spaces are to overwrite left over chars

  delay(100);             // wait for 100 ms (reduces LCD flicker)
}
```

## 5.b. OrangutanBuzzer - High-Level Buzzer Control Library

## Overview

This library allows you to easily control the piezo buzzer on the **3pi robot** [<http://www.pololu.com/catalog/product/975>], **Orangutan SV-168** [<http://www.pololu.com/catalog/product/1225>], and **Orangutan LV-168** [<http://www.pololu.com/catalog/product/775>]. It gives you the option of playing either a note or a frequency for a specified duration at a specified volume, or you can use the **play()** method to play an entire melody in the background. Buzzer control is achieved using one of the Timer1 PWM outputs, and duration timing is performed using a Timer1 overflow interrupt, so **this library will conflict with any other libraries that rely on or reconfigure Timer1**. For example, the Arduino function **analogWrite()** will not work on the Timer1 PWM output pins once you have started to use this library in your sketch.

The benefit to this approach is that you can play notes on the buzzer while leaving the CPU mostly free to execute the rest of your code. This means you can have a melody playing in the background while your Orangutan or 3pi does its main task. You can poll the **isPlaying()** method to determine when the buzzer is finished playing.

You do not need to initialize your OrangutanBuzzer object before use. All initialization is performed automatically when needed.

All of the methods in this class are static; you should never have more than one instance of an OrangutanBuzzer object in your sketch.

## OrangutanBuzzer Methods

Complete documentation of this library's methods can be found in **Section 4** of the **Pololu AVR Library Command Reference** [<http://www.pololu.com/docs/0J18>].

## Usage Examples

This library comes with three example sketches that you can load by going to **File > Sketchbook > Examples > Library-OrangutanBuzzer**.

### 1. OrangutanBuzzerExample

Demonstrates one way to use this library's **playNote()** method to play a simple melody stored in RAM. It should immediately start playing the melody, and you can use the top user pushbutton to stop and replay the melody. The example is structured so that you can add your own code to **loop()** and the melody will still play normally in the background, assuming your code executes quickly enough to avoid inserting delays between the notes. You can use this same technique to play melodies that have been stored in EEPROM (the mega168 has enough room in EEPROM to store 170 notes).

```
#include <OrangutanLCD.h>
#include <OrangutanPushbuttons.h>
#include <OrangutanBuzzer.h>

/*
 * OrangutanBuzzerExample
 *
 * This example uses the OrangutanBuzzer library to play a series of notes on
 * the Orangutan's/3pi's buzzer. It also uses the OrangutanLCD library
 * to display the notes its playing, and it uses the OrangutanPushbuttons
 * library to allow the user to stop/reset the melody with the top
 * pushbutton.
 */

#define MELODY_LENGTH 95

// These arrays take up a total of 285 bytes of RAM (out of a 1k limit)
unsigned char note[MELODY_LENGTH] =
{
  E(5), SILENT_NOTE, E(5), SILENT_NOTE, E(5), SILENT_NOTE, C(5), E(5),
  G(5), SILENT_NOTE, G(4), SILENT_NOTE,

  C(5), G(4), SILENT_NOTE, E(4), A(4), B(4), B_FLAT(4), A(4), G(4),
  E(5), G(5), A(5), F(5), G(5), SILENT_NOTE, E(5), C(5), D(5), B(4),

  C(5), G(4), SILENT_NOTE, E(4), A(4), B(4), B_FLAT(4), A(4), G(4),
  E(5), G(5), A(5), F(5), G(5), SILENT_NOTE, E(5), C(5), D(5), B(4),

  SILENT_NOTE, G(5), F_SHARP(5), F(5), D_SHARP(5), E(5), SILENT_NOTE,
  G_SHARP(4), A(4), C(5), SILENT_NOTE, A(4), C(5), D(5),

  SILENT_NOTE, G(5), F_SHARP(5), F(5), D_SHARP(5), E(5), SILENT_NOTE,
  C(6), SILENT_NOTE, C(6), SILENT_NOTE, C(6),

  SILENT_NOTE, G(5), F_SHARP(5), F(5), D_SHARP(5), E(5), SILENT_NOTE,
  G_SHARP(4), A(4), C(5), SILENT_NOTE, A(4), C(5), D(5),

  SILENT_NOTE, E_FLAT(5), SILENT_NOTE, D(5), C(5)
};

unsigned int duration[MELODY_LENGTH] =
{
  100, 25, 125, 125, 125, 125, 125, 250, 250, 250, 250, 250,

  375, 125, 250, 375, 250, 250, 125, 250, 167, 167, 167, 250, 125, 125,
  125, 250, 125, 125, 375,

  375, 125, 250, 375, 250, 250, 125, 250, 167, 167, 167, 250, 125, 125,
  125, 250, 125, 125, 375,

  250, 125, 125, 125, 250, 125, 125, 125, 125, 125, 125, 125, 125, 125,
  250, 125, 125, 125, 250, 125, 125, 200, 50, 100, 25, 500,

  250, 125, 125, 125, 250, 125, 125, 125, 125, 125, 125, 125, 125, 125,
  250, 250, 125, 375, 500
};
```

```

OrangutanLCD lcd;
OrangutanPushbuttons buttons;
OrangutanBuzzer buzzer;
unsigned char currentIdx;

void setup()                // run once, when the sketch starts
{
  currentIdx = 0;
  lcd.print("Music!");
}

void loop()                 // run over and over again
{
  // if we haven't finished playing the song and
  // the buzzer is ready for the next note, play the next note
  if (currentIdx < MELODY_LENGTH && !buzzer.isPlaying())
  {
    // play note at max volume
    buzzer.playNote(note[currentIdx], duration[currentIdx], 15);

    // optional LCD feedback (for fun)
    lcd.gotoXY(0, 1);           // go to start of the second LCD line
    lcd.print((unsigned int)note[currentIdx]); // print integer value of current note
    lcd.print(" ");           // overwrite any left over characters
    currentIdx++;
  }

  // Insert some other useful code here...
  // the melody will play normally while the rest of your code executes
  // as long as it executes quickly enough to keep from inserting delays
  // between the notes.

  // For example, let the top user pushbutton function as a stop/reset melody button
  if (buttons.isPressed(TOP_BUTTON))
  {
    buzzer.stopPlaying(); // silence the buzzer
    if (currentIdx < MELODY_LENGTH)
      currentIdx = MELODY_LENGTH; // terminate the melody
    else
      currentIdx = 0; // restart the melody
    buttons.waitForRelease(TOP_BUTTON); // wait here for the button to be released
  }
}

```

## 2. OrangutanBuzzerExample2

Demonstrates how you can use this library's **play()** method to start a melody playing. Once started, the melody will play all the way to the end with no further action required from your code, and the rest of your program will execute as normal while the melody plays in the background. The **play()** method is driven entirely by the Timer1 overflow interrupt. The top user pushbutton will play a fugue by Bach from program memory, the middle user pushbutton will quietly play the C major scale up and back down from RAM, and the bottom user pushbutton will stop any melody that is currently playing or play a single note if the buzzer is currently inactive.

```

#include <OrangutanLCD.h>
#include <OrangutanPushbuttons.h>
#include <OrangutanBuzzer.h>

/*
 * OrangutanBuzzerExample2
 *
 * This example uses the OrangutanBuzzer library to play a series of notes on
 * the Orangutan's/3pi's buzzer. It uses the OrangutanPushbuttons
 * library to allow the user select which melody plays.
 *
 * This example demonstrates the use of the OrangutanBuzzer::play() method,
 * which plays the specified melody entirely in the background, requiring
 * no further action from the user once the method is called. The CPU
 * is then free to execute other code while the melody plays.
 */

OrangutanLCD lcd;
OrangutanPushbuttons buttons;

```



```

OrangutanBuzzer buzzer;

#include <avr/pgmspace.h> // this lets us refer to data in program space (i.e. flash)
// store this fugue in program space using the PROGMEM macro.
// Later we will play it directly from program space, bypassing the need to load it
// all into RAM first.
const char fugue[] PROGMEM =
"! O5 L16 agafaea dac+adaea fa<aa<bac#a dac#adaea f"
"O6 dcd<b-d<ad<g d<f+d<gd<ad<b- d<dd<ed<f+d<g d<f+d<gd<ad"
"L8 MS <b-d<b-d MLe-<ge-<g MSc<ac<a ML d<fd<f O5 MS b-gb-g"
"ML >c#e>c#e MS afaf ML gc#gc# MS fdfd ML e<b-e<b-"
"O6 L16ragafaea dac#adaea fa<aa<bac#a dac#adaea faeadaca"
"<b-acadg<b-g egdgcg<b-g <ag<b-gcf<af dcf<b-f<af"
"<gf<af<b-e<ge c#e<b-e<ae<ge <fe<ge<ad<fd"
"O5 e>ee>ef>df>d b->c#b->c#a>df>d e>ee>ef>df>d"
"e>d>c#>db>d>c#b >c#agaegfe f O6 dc#dfdc#<b c#4";

void setup() // run once, when the sketch starts
{
  lcd.print("Press a");
  lcd.gotoXY(0, 1);
  lcd.print("button..");
}

void loop() // run over and over again
{
  // wait here for one of the three buttons to be pushed
  unsigned char button = buttons.waitForButton(ALL_BUTTONS);
  lcd.clear();

  if (button == TOP_BUTTON)
  {
    buzzer.playFromProgramSpace(fugue);

    lcd.print("Fugue!");
    lcd.gotoXY(0, 1);
    lcd.print("flash ->");
  }
  if (button == MIDDLE_BUTTON)
  {
    buzzer.play("! V8 cdefgab>cbagfedc");
    lcd.print("C Major");
    lcd.gotoXY(0, 1);
    lcd.print("RAM ->");
  }
  if (button == BOTTOM_BUTTON)
  {
    if (buzzer.isPlaying())
    {
      buzzer.stopPlaying();
      lcd.print("stopped");
    }
    else
    {
      buzzer.playNote(A(5), 200, 15);
      lcd.print("note A5");
    }
  }
}

```

### 3. OrangutanBuzzerExample3

Demonstrates the use of this library's `playMode()` and `playCheck()` methods. In this example, automatic play mode is used to allow the melody to keep playing while it blinks the red user LED. Then the mode is switched to play-check mode during a phase where we are trying to accurately measure time. There are three `#define` macros that allow you to run this example in different ways and observe the result. Please see the comments at the top of the sketch for more detailed information.

```

#include <OrangutanLCD.h>
#include <OrangutanLEDs.h>
#include <OrangutanBuzzer.h>

/*
 * OrangutanBuzzerExample3

```

```

*
* This example uses the OrangutanBuzzer library to play a series of notes on
* the target's piezo buzzer.
*
* This example demonstrates the use of the OrangutanBuzzer::playMode()
* and OrangutanBuzzer::playCheck() methods, which allow you to select
* whether the melody sequence initiated by OrangutanBuzzer::play() is
* played automatically in the background by the Timer1 interrupt, or if
* the play is driven by the playCheck() method in your main loop.
*
* Automatic play mode should be used if your code has a lot of delays
* and is not time critical. In this example, automatic mode is used
* to allow the melody to keep playing while we blink the red user LED.
*
* Play-check mode should be used during parts of your code that are
* time critical. In automatic mode, the Timer1 interrupt is very slow
* when it loads the next note, and this can delay the execution of your.
* Using play-check mode allows you to control when the next note is
* loaded so that it doesn't occur in the middle of some time-sensitive
* measurement. In our example we use play-check mode to keep the melody
* going while performing timing measurements using Timer2. After the
* measurements, the maximum time measured is displayed on the LCD.
*
* Immediately below are three #define statements that allow you to alter
* the way this program runs. You should have one of the three lines
* uncommented while commenting out the other two:
*
* If only WORKING_CORRECTLY is uncommented, the program should run in its
* ideal state, using automatic play mode during the LED-blinking phase
* and using play-check mode during the timing phase. The maximum recorded
* time should be 20, as expected.
*
* If only ALWAYS_AUTOMATIC is uncommented, the program will use automatic
* play mode during both the LED-blinking phase and the timing phase. Here
* you will see the effect this has on the time measurements (instead of 20,
* you should see a maximum reading of around 27 or 28).
*
* If only ALWAYS_CHECK is uncommented, the program will be in play-check
* mode during both the LED-blinking phase and the timing phase. Here you
* will see the effect that the LED-blinking delays have on play-check
* mode (the sequence will be very choppy while the LED is blinking, but
* sound normal during the timing phase). The maximum timing reading should
* be 20, as expected.
*/

// *** UNCOMMENT ONE OF THE FOLLOWING PRECOMPILER DIRECTIVES ***
// (the remaining two should be commented out)
#define WORKING_CORRECTLY // this is the right way to use playMode()
// #define ALWAYS_AUTOMATIC // playMode() is always PLAY_AUTOMATIC (timing is inaccurate)
// #define ALWAYS_CHECK // playMode() is always PLAY_CHECK (delays interrupt the sequence)

OrangutanLEDS leds;
OrangutanBuzzer buzzer;
OrangutanLCD lcd;

#include <avr/pgmspace.h>
const char rhapsody[] PROGMEM = "06 T40 L16 d#<b#<f#<d#<f#<bd#f#"
    "T80 c#<b-<f#<c#<f#<b-c#8"
    "T180 d#<f#<d#f#>bd#f#c#b-<f#c#f#>b-c#8 c>c#<c#>c#<b>c#<c#>c#<c#>c#<b>c#<c#>c#"
    "c>c#<c#>c#<b->c#<c#>c#<c#>c#<b->c#<c#>c#"
    "c>c#<c#>c#f>c#<c#>c#c>c#<c#>c#f>c#<c#>c#"
    "c>c#<c#>c#f#>c#<c#>c#c>c#<c#>c#f#>c#<c#>c#d#bb-bd#bf#d#c#b-ab-c#b-f#d#";

void setup() // run once, when the sketch starts
{
    TCCR2A = 0; // configure timer2 to run at 78 kHz
    TCCR2B = 0x06; // and overflow when TCNT2 = 256 (~3 ms)
    buzzer.playFromProgramSpace(rhapsody);
}

void loop() // run over and over again
{
    // allow the sequence to keep playing automatically through the following delays
    #ifndef ALWAYS_CHECK
        buzzer.playMode(PLAY_AUTOMATIC);
    #else
        buzzer.playMode(PLAY_CHECK);
    #endif
}

```

```

#endif
  lcd.gotoXY(0, 0);
  lcd.print("blink!");
  int i;
  for (i = 0; i < 8; i++)
  {
#ifdef ALWAYS_CHECK
  buzzer.playCheck();
#endif
  leds.red(HIGH);
  delay(500);
  leds.red(LOW);
  delay(500);
}

  lcd.gotoXY(0, 0);
  lcd.print("timing");
  lcd.gotoXY(0, 1);
  lcd.print(" "); // clear bottom LCD line
  // turn off automatic playing so that our time-critical code won't be interrupted by
  // the buzzer's long timer1 interrupt. Otherwise, this interrupt could throw off our
  // timing measurements. Instead, we will now use playCheck() to keep the sequence
  // playing in a way that won't throw off our measurements.
#ifdef ALWAYS_AUTOMATIC
  buzzer.playMode(PLAY_CHECK);
#endif
  unsigned char maxTime = 0;
  for (i = 0; i < 8000; i++)
  {
    TCNT2 = 0;
    while (TCNT2 < 20) // time for ~250 us
      ;
    if (TCNT2 > maxTime)
      maxTime = TCNT2; // if the elapsed time is greater than the previous max, save it
#ifdef ALWAYS_AUTOMATIC
    buzzer.playCheck(); // check if it's time to play the next note and play it if so
#endif
  }
  lcd.gotoXY(0, 1);
  lcd.print("max=");
  lcd.print((unsigned int)maxTime);
  lcd.print(' '); // overwrite any left over characters
}

```

### 5.c. OrangutanLCD - LCD Control Library

## Overview

This library gives you the ability to control the 8×2 character LCD on the **3pi robot** [<http://www.pololu.com/catalog/product/975>], **Orangutan SV-168** [<http://www.pololu.com/catalog/product/1225>], and **Orangutan LV-168** [<http://www.pololu.com/catalog/product/775>]. It implements the standard 4-bit HD44780 protocol, and it uses the busy-wait-flag feature to avoid the unnecessarily long delays present in other 4-bit LCD Arduino libraries. This comprehensive library is meant to offer as much LCD control as possible, so it most likely gives you more methods than you need. You can comment out unneeded methods (e.g. `showCursor()`) in `OrangutanLCD.cpp` to make this library smaller. If you do this, remember to delete `OrangutanLCD.o` so and restart the Arduino IDE so that the library will be recompiled.

This library is designed to gracefully handle alternate use of the four LCD data lines. It will change their data direction registers and output states only when needed for an LCD command, after which it will immediately restore the registers to their previous states. This allows the LCD data lines to function as pushbutton inputs and an LED driver on the Orangutan and 3pi.

You do not need to initialize your `OrangutanLCD` object before use. All initialization is performed automatically when needed.

All of the methods in this class are static; you should never have more than one instance of an `OrangutanLCD` object in your sketch.

## OrangutanLCD Methods

Complete documentation of this library's methods can be found in **Section 5** of the **Pololu AVR Library Command Reference** [<http://www.pololu.com/docs/0J18>].

## Usage Examples

This library comes with two example sketches that you can load by going to **File > Sketchbook > Examples > Library-OrangutanLCD**. Note that most of the other libraries have example sketches that use the LCD, so please see these for more OrangutanLCD usage examples.

### 1. OrangutanLCDExample

Demonstrates shifting the contents of the display by moving the word “Hello” around the two lines of the LCD.

```
#include <OrangutanLCD.h>

/*
 * OrangutanLCDExample
 *
 * This example uses the OrangutanLCD library to write "Hello"
 * on the LCD and then move it around the display area.
 */

OrangutanLCD lcd;

void setup()                // run once, when the sketch starts
{
}

void loop()                 // run over and over again
{
  lcd.print("Hello");      // display "Hello" at (0, 0), a.k.a. upper-left
  delay(200);

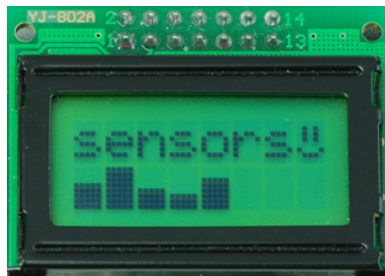
  // shift the display right every 200ms three times
  lcd.scroll(LCD_RIGHT, 3, 200);
  lcd.clear();             // clear the LCD
  lcd.gotoXY(3, 1);       // go to the fourth character of the second LCD line
  lcd.print("Hello");     // display "Hello" at (3, 1), a.k.a. lower-right
  delay(200);

  // shift the display left every 200ms three times
  lcd.scroll(LCD_LEFT, 3, 200);
  lcd.clear();            // clear the LCD
}

```

### 1. OrangutanLCDExample2

Demonstrates creating and displaying custom characters on the LCD. The following picture shows an example of custom characters, using them to display a bar graph of sensor readings and a smiley face:



```
#include <OrangutanPushbuttons.h>
#include <OrangutanLCD.h>
#include <stdlib.h>         // used for its "random" and "srandom" functions

/*
 * OrangutanLCDExample2
 *
 * This example uses the OrangutanLCD library to display custom characters
 * on the LCD.  Simply push a any user pushbutton to display a new,

```

```
* randomly chosen, custom mood character.
*/

OrangutanLCD lcd;

// define some custom "mood" characters
#include <avr/pgmspace.h> // this lets us refer to data in program space (i.e. flash)
const char happy[] PROGMEM = {
  0b00000, // the 5 bits that make up the top row of the 5x8 character
  0b01010,
  0b01010,
  0b01010,
  0b00000,
  0b10001,
  0b01110,
  0b00000
};

const char sad[] PROGMEM = {
  0b00000,
  0b01010,
  0b01010,
  0b01010,
  0b00000,
  0b01110,
  0b10001,
  0b00000
};

const char indifferent[] PROGMEM = {
  0b00000,
  0b01010,
  0b01010,
  0b01010,
  0b00000,
  0b00000,
  0b01110,
  0b00000
};

const char surprised[] PROGMEM = {
  0b00000,
  0b01010,
  0b01010,
  0b00000,
  0b01110,
  0b10001,
  0b10001,
  0b01110
};

const char mocking[] PROGMEM = {
  0b00000,
  0b01010,
  0b01010,
  0b01010,
  0b00000,
  0b11111,
  0b00101,
  0b00010
};

char prevMood = 5;

void setup() // run once, when the sketch starts
{
  lcd.loadCustomCharacter(happy, 0);
  lcd.loadCustomCharacter(sad, 1);
  lcd.loadCustomCharacter(indifferent, 2);
  lcd.loadCustomCharacter(surprised, 3);
  lcd.loadCustomCharacter(mocking, 4);
  lcd.clear(); // must be called before we can use the custom chars
  lcd.print("mood: ?");

  // initialize the random number generator based on how long we hold the button the first time
  OrangutanPushbuttons::waitForPress(ALL_BUTTONS);
  long seed = 0;
}
```

```
    while (OrangutanPushbuttons::isPressed(ALL_BUTTONS))
        seed++;
    random(seed);          // the same as: randomSeed((unsigned int)seed);
}

void loop()              // run over and over again
{
    lcd.gotoXY(6, 0);    // move cursor to the correct position

    // ensure we get a new mood that differs from the previous
    char mood;
    do
    {
        mood = random()%5;    // the same as: mood = random(5);
    } while (mood == prevMood);
    prevMood = mood;

    lcd.print(mood);     // print a random mood character

    // wait for any button to be pressed
    OrangutanPushbuttons::waitForButton(ALL_BUTTONS);
}
```

#### 5.d. OrangutanLEDs - LED Control Library



## Overview

This library allows you to easily control the LED(s) on the **3pi robot** [<http://www.pololu.com/catalog/product/975>], **Orangutan SV-168** [<http://www.pololu.com/catalog/product/1225>], **Orangutan LV-168** [<http://www.pololu.com/catalog/product/775>], and **Baby Orangutan B** [<http://www.pololu.com/catalog/product/1216>]. The 3pi, Orangutan SV-168, and Orangutan LV-168 have two user LEDs: a red LED on the bottom left and a green LED on the top right. On the Orangutan SV-168 and LV-168, the LEDs are on the top side of the PCB with the red LED on the bottom left and the green LED on the top right. On the 3pi, the LEDs are on the bottom side of the PCB with the red LED on the right (when looking at the bottom) and the green LED on the left. There are through-holes on the 3pi PCB that allow you to solder your own parallel LEDs to the top side of the board. Note that the green LED on the 3pi and the Orangutans is on a line shared by the LCD, so it will often faintly flicker whenever your program updates the LCD. The Baby Orangutan has a single, red user LED.

You do not need to initialize your `OrangutanLEDs` object before use. All initialization is performed automatically when needed.

All of the methods in this class are static; you should never have more than one instance of an `OrangutanLEDs` object in your sketch.

## OrangutanLEDs Methods

Complete documentation of this library's methods can be found in **Section 6** of the **Pololu AVR Library Command Reference** [<http://www.pololu.com/docs/0J18>].

## Usage Examples

This library comes with an example sketch that you can load by going to **File > Sketchbook > Examples > Library-OrangutanLEDS**.

### 1. OrangutanLEDExample

Alternately blinks the red and green LEDs on the Orangutan sV-168, Orangutan LV-168, or 3pi robot. If you run this program on the Baby Orangutan B, you will only see the red user LED blink, but you can connect an external LED to pin PD7 (Arduino pin 7) if you want to see the second LED blink. If you do this, don't forget to include a current-limiting resistor!

```
#include <OrangutanLEDS.h>

/*
 * OrangutanLEDExample
 *
 * This sketch uses the OrangutanLEDS library to control the red and green
 * LEDs on the Orangutan SV-168, LV-168, or 3pi. It will also work to control the
 * red LED on the Baby Orangutan B (which lacks a second, green LED).
 */

OrangutanLEDS leds;

void setup()                // run once, when the sketch starts
{
}

void loop()                 // run over and over again
{
  leds.red(HIGH);           // red LED on
  delay(1000);              // waits for a second
  leds.red(LOW);            // red LED off
  delay(1000);              // waits for a second

  // the following segment will not work on the Baby Orangutan
  leds.green(HIGH);         // green LED on
  delay(500);               // waits for 0.5 seconds
  leds.green(LOW);          // green LED off
  delay(500);               // waits for 0.5 seconds
}
```

### 5.e. OrangutanMotors - Motor Control Library

## Overview

This library gives you the ability to control the motor drivers on the **3pi robot** [<http://www.pololu.com/catalog/product/975>], **Orangutan SV-168** [<http://www.pololu.com/catalog/product/1225>], **Orangutan LV-168** [<http://www.pololu.com/catalog/product/775>], and **Baby Orangutan B** [<http://www.pololu.com/catalog/product/1216>]. It accomplishes this by using the four hardware PWM outputs from timers Timer0 and Timer2, so **this library will conflict with any other libraries that rely on or reconfigure Timer0 or Timer2**. Unfortunately the Arduino environment relies on Timer0 for its millis() and delay() functions, but this library enables a Timer2 interrupt that restores the functionality of millis() and delay() to normal.

You do not need to initialize your OrangutanMotors object before use. All initialization is performed automatically when needed.

All of the methods in this class are static; you should never have more than one instance of an OrangutanMotors object in your sketch.

## **OrangutanMotors Methods**

Complete documentation of this library's methods can be found in **Section 7** of the **Pololu AVR Library Command Reference** [<http://www.pololu.com/docs/0J18>].

## Usage Examples

This library comes with two example sketches that you can load by going to **File > Sketchbook > Examples > Library-OrangutanMotors**.

### 1. OrangutanMotorExample

Demonstrates controlling the motors using the trimmer potentiometer and uses the red LED for feedback. This example is designed to work on the 3pi robot, Orangutan SV-168, Orangutan LV-168, and Baby Orangutan B.

```
#include <OrangutanLEDs.h>
#include <OrangutanAnalog.h>
#include <OrangutanMotors.h>

/*
 * OrangutanMotorExample
 *
 * This example uses the OrangutanMotors library to drive
 * motors in response to the position of user trimmer potentiometer
 * and blinks the red user LED at a rate determined by the trimmer
 * potentiometer position. It uses the OrangutanAnalog library to measure
 * the trimpot position, and it uses the OrangutanLEDs library to provide
 * limited feedback with the red and green user LEDs.
 */

OrangutanAnalog analog;
OrangutanLEDs leds;
OrangutanMotors motors;
unsigned long prevMillis = 0;

void setup()           // run once, when the sketch starts
{
}

void loop()            // run over and over again
{
  // note that the following line could also be accomplished with:
  // int pot = analogRead(7);
  int pot = analog.readTrimpot();    // determine the trimpot position

  // turn pot reading into number between -256 and 256
  int motorSpeed = (512 - pot) / 2;
  motors.setSpeeds(motorSpeed, motorSpeed);

  // the following code blinks the red user LED at a rate that's approximately
  // inversely proportional to the position of the user trim pot
  if (millis() - prevMillis >= pot)
  {
    prevMillis = millis() + 100;
    leds.red(HIGH);    // turn red LED on
  }
  delay(100);
  leds.red(LOW);      // turn red LED off
}
```

### 2. OrangutanMotorExample2

Demonstrates controlling the motors using the trimmer potentiometer, but it uses the LCD for most of the feedback, so it will only fully work on the Orangutan SV-168, Orangutan LV-168, and 3pi robot.

```
#include <OrangutanLEDs.h>
#include <OrangutanAnalog.h>
#include <OrangutanMotors.h>
#include <OrangutanLCD.h>

/*
 * OrangutanMotorExample2
 *
 * This example uses the OrangutanMotors and OrangutanLCD libraries to
 * drive motors in response to the position of user trimmer potentiometer
 */
```

```
* and to display the potentiometer position and desired motor speed
* on the LCD. It uses the OrangutanAnalog library to measure the
* trimpot position, and it uses the OrangutanLEDs library to provide
* limited feedback with the red and green user LEDs.
*/

OrangutanLCD lcd;
OrangutanMotors motors;
OrangutanAnalog analog;
OrangutanLEDs leds;

void setup()                // run once, when the sketch starts
{
}

void loop()                 // run over and over again
{
  // note that the following line could also be accomplished with:
  // int pot = analogRead(7);
  int pot = analog.readTrimpot(); // determine the trimpot position

  // avoid clearing the LCD to reduce flicker
  lcd.gotoXY(0, 0);
  lcd.print("pot=");
  lcd.print(pot);           // print the trim pot position (0 - 1023)
  lcd.print(" ");          // overwrite any left over digits

  int motorSpeed = (512 - pot) / 2;
  lcd.gotoXY(0, 1);
  lcd.print("spd=");
  lcd.print(motorSpeed);    // print the resulting motor speed (-255 - 256)
  lcd.print(" ");
  motors.setSpeeds(motorSpeed, motorSpeed); // set speeds of motors 1 and 2

  // all LEDs off
  leds.red(LOW);
  leds.green(LOW);
  // turn green LED on when motors are spinning forward
  if (motorSpeed > 0)
    leds.green(HIGH);
  // turn red LED on when motors are spinning in reverse
  if (motorSpeed < 0)
    leds.red(HIGH);
  delay(100);
}
```

## 5.f. OrangutanPushbuttons - Pushbutton Interface Library

## Overview

This library allows you to easily interface with the three user pushbuttons on the **3pi robot** [<http://www.pololu.com/catalog/product/975>], **Orangutan SV-168** [<http://www.pololu.com/catalog/product/1225>], and **Orangutan LV-168** [<http://www.pololu.com/catalog/product/775>] by either polling for the state of specific buttons or by waiting for press/release events on specifiable buttons. The **waitFor\_\_\_\_()** methods in this library automatically take care of button debouncing.

You do not need to initialize your OrangutanPushbuttons object before use. All initialization is performed automatically when needed.

All of the methods in this class are static; you should never have more than one instance of an OrangutanPushbuttons object in your sketch.



## **OrangutanPushbuttons Methods**

Complete documentation of this library's methods can be found in **Section 8** of the **Pololu AVR Library Command Reference** [<http://www.pololu.com/docs/0J18>].

## Usage Examples

This library comes with an example sketch that you can load by going to **File > Sketchbook > Examples > Library-OrangutanPushbuttons**.

### 1. OrangutanPushbuttonExample

Demonstrates interfacing with the user pushbuttons. It will wait for you to push either the top button or the bottom button, at which point it will display on the LCD which button was pressed. It will also detect when that button is subsequently released and display that to the LCD.

```
#include <OrangutanLCD.h>
#include <OrangutanPushbuttons.h>

/*
 * OrangutanPushbuttonExample
 *
 * This example uses the OrangutanPushbuttons library to detect user input from the
 * pushbuttons, and it uses the OrangutanLCD library to display feedback on the LCD.
 */

OrangutanPushbuttons buttons;
OrangutanLCD lcd;

void setup()                                // run once, when the sketch starts
{
}

void loop()                                  // run over and over again
{
  lcd.clear();
  lcd.print("Waiting");

  // wait for either the top or bottom buttons to be pressed
  // store the value of the pressed button in the variable 'button'
  unsigned char button = buttons.waitForPress(TOP_BUTTON | BOTTOM_BUTTON);
  lcd.clear();
  if (button == TOP_BUTTON)                  // display the button that was pressed
    lcd.print("top down");
  else
    lcd.print("bot down");
  buttons.waitForRelease(button);           // wait for that button to be released
  lcd.clear();
  lcd.print("released");                    // display that the button was released
  delay(1000);
}
```

### 5.g. Pololu3pi - Sensor Library for the 3pi Robot

## Overview

This library allows you to easily interface with the five infrared reflectance sensors on the **3pi robot** [<http://www.pololu.com/catalog/product/975>]. Note that in order to use this library, **you must also include PololuQTRSensors.h in your sketch**. You should have something like the following at the top of your sketch:

```
#include <Pololu3pi.h> // gives access to sensor interface functions
#include <PololuQTRSensors.h> // used by Pololu3pi.h
#include <OrangutanMotors.h> // gives access to motor control functions
#include <OrangutanBuzzer.h> // gives access to buzzer control functions
```

Unlike the other Orangutan libraries, you must explicitly call the **init()** method to initialize your Pololu3pi object before using it.

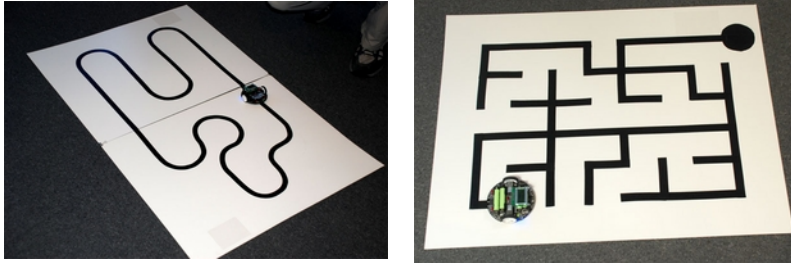
All of the methods in this class are static; you should never have more than one instance of a Pololu3pi object in your sketch.

## Pololu3pi Methods

Complete documentation of this library's methods can be found in **Section 12** of the **Pololu AVR Library Command Reference** [<http://www.pololu.com/docs/0J18>].

## Usage Examples

This library comes with three example sketches that you can load by going to **File > Sketchbook > Examples > Library-Pololu3pi**. There is a simple line-following example, a more advanced PID-based line-following example, and a basic maze-solving example.



For more information on creating line and maze courses, as well as a detailed explanation of the C versions of the demo programs, please see sections 6 and 7 of the **3pi User's Guide** [<http://www.pololu.com/docs/0J21>].