

# Experimenter's Guide for Netduino



(NEDX)  
netduino



## A Few Words

### ABOUT THIS KIT

The overall goal of this kit is fun. Beyond this, the aim is to get you comfortable using a wide range of electronic components through small, simple and easy circuits. The focus is to get each circuit working then giving you the tools to figure out why. If you encounter any problems, want to ask a question, or would like to know more about any part, extra help is only an e-mail away [help@oomlout.com](mailto:help@oomlout.com).



### ABOUT OPEN SOURCE HARDWARE

All of the projects at [.:oomlout:.](http://www.oomlout.com) are open source. What does this mean? It means everything involved in making this kit, be it this guide, 3D models, or code is available for free download. But it goes further, you're also free to reproduce and modify any of this material, then distribute it for yourself. The catch? Quite simple, it is released under a Creative Commons (By - Share Alike) license. This means you must credit [.:oomlout:.](http://www.oomlout.com) in your design and share your developments in a similar manner. Why? We grew up learning and playing with open source software and the experience was good fun, we think it would be lovely if a similar experience was possible with physical things.

More details on the Creative Commons CC (By - Share Alike) License can be found at <http://nedx.org/CCLI>

### ABOUT .: OOMLOUT :.

We're a plucky little design company focusing on producing  
"delightfully fun open source products"

To check out what we are up to

<http://www.oomlout.com>

### ABOUT NETDUINO

A fun little board programable in .NET with limitless potential. Accompanied by a great online community offering support and encouragement.

<http://netduino.com/>

### ABOUT PROBLEMS

We strive to deliver the highest level of quality in each and every thing we produce. If you ever find an ambiguous instruction, a missing piece, or would just like to ask a question, we'll try our best to help out.

[help@oomlout.com](mailto:help@oomlout.com)

(we like hearing about problems it helps us improve future versions)

Thanks For Choosing [.:oomlout:.](http://www.oomlout.com)  
and netduino

Before We Start

{ASEM}	Assembling the Pieces	02
{INST}	Installing the Software	03
{PROG}	A Small Programming Primer	04
{ELEC}	A Small Electronics Primer	06
{FIRS}	Your First Program	08
{HWDD}	Hardware Definitions	09

The Circuits

{NCIR01}	Getting Started - (Blinking LED)	10
{NCIR02}	8 LED Fun - (Multiple LEDs)	12
{NCIR03}	Spin Motor Spin - (Transistor and Motor)	14
{NCIR04}	A Single Servo - (Servos)	16
{NCIR05}	8 More LEDs - (74HC595 Shift Register)	18
{NCIR06}	Music - (Piezo Elements)	20
{NCIR07}	Button Pressing - (Pushbuttons)	22
{NCIR08}	Twisting - (Potentiometers)	24
{NCIR09}	Light - (Photo Resistors)	26
{NCIR10}	Temperature - (TMP36 Temperature Sensor)	28
{NCIR11}	Larger Loads - (Relays)	30

# 01 ASEM

assembling the pieces

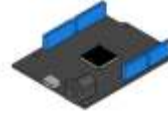
.: PUTTING IT TOGETHER :.



Netduino Holder  
x1



Breadboard  
x1



Netduino  
x1



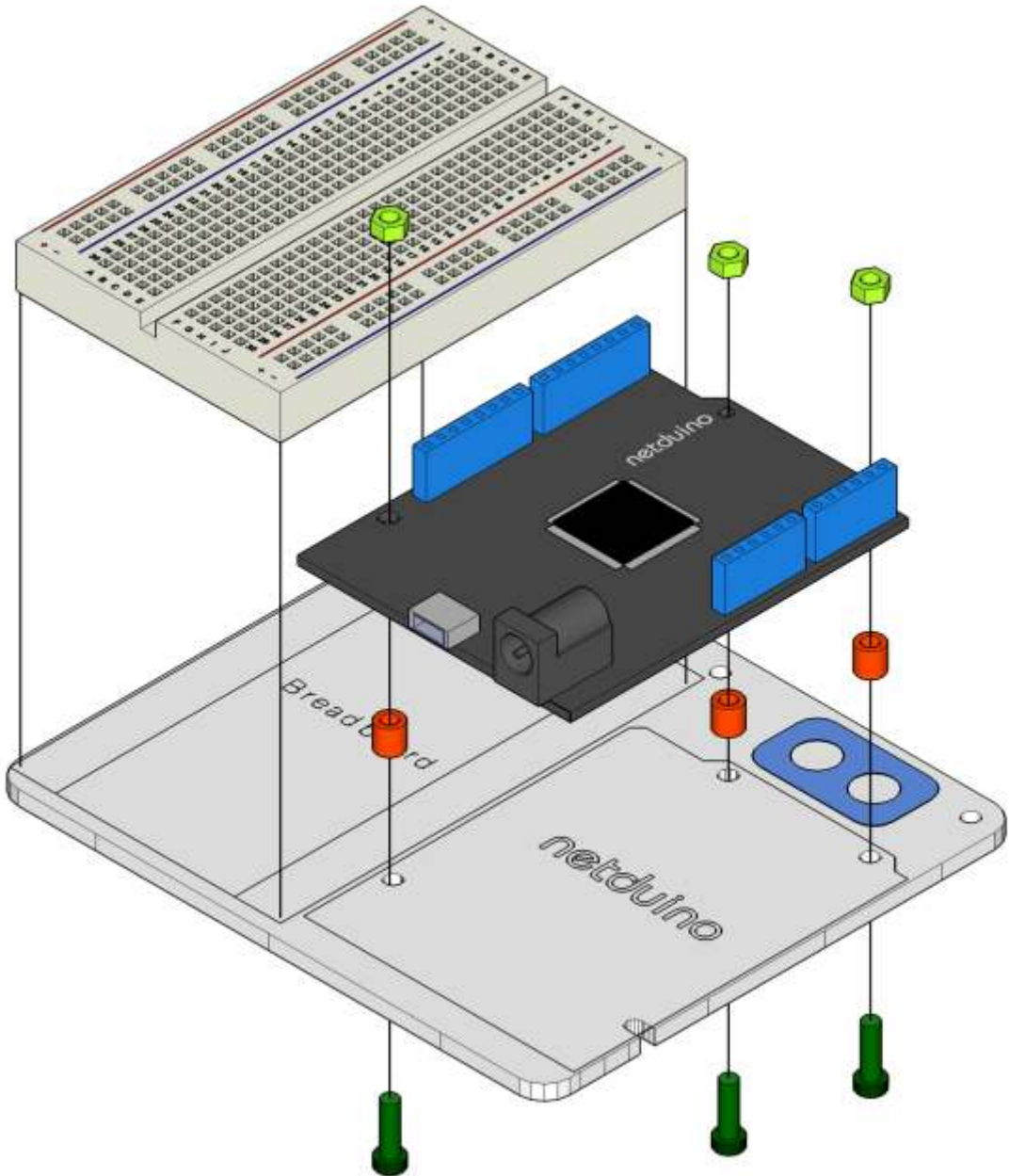
3mm x 10mm bolt  
x3



3mm nut  
x3



3mm spacer  
x3



.: For an introduction to what an Netduino is, visit :.  
.: <http://nedx.org/INTR> :.

## .: INSTALLING THE IDE :.

**02 INST**  
installing  
(software and hardware)

This is the program used to write code for the Netduino.  
The Netduino uses Microsoft's .NET Micro Framework so there are  
a couple of different applications that need installing  
.:extra details can be found online:.  
<http://www.nedx.org/DOWN>

Windows XP, Vista, or Seven

### Step 1: Download & Install Microsoft Visual C# Express 2010

Go to  
<http://nedx.org/MVIS>

download size: ~160 MB  
installed size: ~2.4 GB  
time: ~20 min

### Step 2: Restart If prompted restart

### Step 3: Download & Install .NET Micro Framework SDK

Go to  
<http://nedx.org/NEMF>

download size: ~19 MB  
time: ~10 min

### Step 4: Download & Install Netduino SDK

For 32 bit version      For 64 bit version  
Go to                      Go to  
<http://nedx.org/NS32>    <http://nedx.org/NS64>

download size: ~2 MB  
time: ~5 min

### .:While You Wait:.

Check out what others are doing with their Netduino <http://nedx/PROJ> (projects)  
See what people are saying about Netduinos <http://nedx.org/FORU> (forums)  
Netduino is open source so feel free to  
check out the source files <http://nedx/DOWN> (downloads)

.: NOTE: :.  
.: Encountering problems? :.  
.: Would like more details, want to ask a question :.  
.: <http://nedx.org/FORU> :.

### NETDUINO PROGRAMMING IN BRIEF

The Netduino is programmed in the C# language. This is a quick little primer targeted at people who have a little bit of programming experience and just need a briefing on the idiosyncracies of C#. If you find the concepts a bit daunting, don't worry, you can start going through the circuits and pick up most of it along the way. For a more in-depth intro, the Netduino forums are a great resource. <http://nedx.org/FORU>

### STRUCTURE

Each Netduino program has the same structure.

Libraries - At the top of each program these determine what functions can be called.  
namespace - Contains within it all the code and declarations for our program  
public class Program - Contains all our programs methods  
public static void Main() - The function run on power up  
while (true) - A loop that executes until power is removed (optional but used throughout this guide)

### SYNTAX

The C# language has quite strict formatting requirements. But don't worry the IDE often auto corrects for you or will generate a helpful error code telling you right where the problem is.

// (single line comment)  
It is often useful to write notes to yourself as you go along about what each line of code does. To do this type two forward slashes and everything until the end of the line will be ignored by your program.

/\* \*/(multi line comment)  
If you have a lot to say you can span several lines as a comment. Everything between these two symbols will be ignored in your program.

{ } (curly brackets)  
Used to define when a block of code starts and ends (used in functions as well as loops).

;(semicolon)  
Each line of code must be ended with a semicolon (a missing semicolon is often the reason for a program refusing to compile).

### VARIABLES

A program is nothing more than instructions to move numbers around in an intelligent way. Variables are used to do the moving.

int (integer)  
The main workhorse, stores a number with no decimal places. Often used when counting things or storing program constants

bool (boolean)  
A simple True or False variable. Useful for logical comparisons.

float (float)  
Used for floating point math (decimals).

char (character)  
Stores one character using the ASCII code (ie 'A' = 65). The Netduino handles strings as an array of char's.

## MATH OPERATORS

Operators used for manipulating numbers. (they work like simple math).

= (assignment) makes something equal to something else (eg.  $x = 10 * 2$  (x now equals 20))  
% (modulo) gives the remainder when one number is divided by another (ex.  $12 \% 10$  (gives 2))  
+ (addition)  
- (subtraction)  
\* (multiplication)  
/ (division)

## COMPARISON OPERATORS

Operators used for logical comparison.

== (equal to) (eg.  $12 == 10$  is FALSE or  $12 == 12$  is TRUE)  
!= (not equal to) (eg.  $12 != 10$  is TRUE or  $12 != 12$  is FALSE)  
< (less than) (eg.  $12 < 10$  is FALSE or  $12 < 12$  is FALSE or  $12 < 14$  is TRUE)  
> (greater than) (eg.  $12 > 10$  is TRUE or  $12 > 12$  is FALSE or  $12 > 14$  is FALSE)

## CONTROL STRUCTURE

Programs are reliant on controlling what runs next, here are the basic control elements.

```
if(condition){  
else if( condition ){  
else { }
```

This will execute the code between the curly brackets if the condition is true, and if not it will test the else if condition if that is also false the else code will execute.

```
for(int i = 0; i < #repeats; i++){  
}
```

Used when you would like to repeat a chunk of code a number of times (can count up i++ or down i-- or use any variable)

## DIGITAL (Pi ns. GPIO\_PIN\_D0-D13)

### Output

```
outp = new OutputPort(Pi ns. GPIO_PIN_D#, false);  
Used to control a pins state either high (true 3.3v) or low (false 0v)  
ex: outp. Write(true or false);
```

### Input

```
inp = new InputPort(Pi ns. GPIO_PIN_D#, false,  
Port. ResistorMode. Disabled);  
Used to read a pins state either high (true >-1.6v) or low (false <-1.6v)  
ex: inp. Read();
```

## ANALOG

### Output (Pi ns. GPIO\_PIN\_D5, D6, D9, D10)

```
pwmp = new PWM(Pi ns. GPIO_PIN_D#);  
Used to turn the pin on and off very fast mimicking an analog voltage.  
pwmp. SetDutyCycle(0-100%);
```

### Input (Pi ns. GPIO\_PIN\_A0-A5)

```
anap = new AnalogInput(Pi ns. GPIO_PIN_A#);  
Used to read the voltage present on a pin  
anap. Read();  
range 0-1023 (10 bit) (0-3.3v)
```

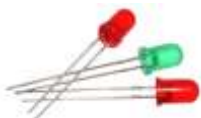
## ELECTRONICS IN BRIEF

No previous electronic experience is required to have fun with this kit. Here are a few details about each component to make identifying, and perhaps understanding them, a bit easier. If at any point you are worried about how a component is used or why it's not working the internet offers a treasure trove of advice, or we can be contacted at [help@oomlout.com](mailto:help@oomlout.com)

## COMPONENT DETAILS

### LED

(Light Emitting Diode)



**What it Does:**

Emits light when a small current is passed through it. (only in one direction)

**Identifying:**

Looks like a mini light bulb.

**No. of Leads:**

2 (one longer, this one connects to positive)

**Things to watch out for:**

- Will only work in one direction
- Requires a current limiting resistor

**More Details:**

<http://nedx.org/LED>

### Diode



**What it Does:**

The electronic equivalent of a one way valve. Allowing current to flow in one direction but not the other.

**Identifying:**

Usually a cylinder with wires extending from either end. (and an off center line indicating polarity)

**No. of Leads:**

2

**Things to watch out for:**

- Will only work in one direction (current will flow if end with the line is connected to ground)

**More Details:**

<http://nedx.org/DIOD>

### Resistors



**What it Does:**

Restricts the amount of current that can flow through a circuit.

**Identifying:**

Cylinder with wires extending from either end. The value is displayed using a color coding system (for details see next page)

**No. of Leads:**

2

**Things to watch out for:**

- Easy to grab the wrong value (double check the colors before using)

**More Details:**

<http://nedx.org/RESI>

### Transistor



**What it Does:**

Uses a small current to switch or amplify a much larger current.

**Identifying:**

Comes in many different packages but you can read the part number off the package. (P2N2222AG in this kit and find a datasheet online)

**No. of Leads:**

3 (Base, Collector, Emitter)

**Things to watch out for:**

- Plugging in the right way round (also a current limiting resistor is often needed on the base)

**More Details:**

<http://nedx.org/TRAN>

### Hobby Servo



**What it Does:**

Takes a timed pulse and converts it into an angular position of the output shaft.

**Identifying:**

A plastic box with 3 wires coming out one side and a shaft with a plastic horn out the top.

**No. of Leads:**

3

**Things to watch out for:**

- The plug is not polarized so make sure it is plugged in the right way.

**More Details:**

<http://nedx.org/SERV>

### DC Motor



**What it Does:**

Spins when a current is passed through it.

**Identifying:**

This one is easy, it looks like a motor.

Usually a cylinder with a shaft coming out of one end.

**No. of Leads:**

2

**Things to watch out for:**

- Using a transistor or relay that is rated for the size of motor you're using.

**More Details:**

<http://nedx.org/MOTO>

## COMPONENT DETAILS (CONT.)

### Piezo Element



**What it Does:**  
A pulse of current will cause it to click.  
A stream of pulses will cause it to emit a tone.

**Identifying:**  
In this kit it comes in a little black barrel, but sometimes they are just a gold disc.

**No. of Leads:**  
2

**Things to watch out for:**  
- Difficult to misuse.

**More Details:**  
<http://nedx.org/PIEZ>

### IC (Integrated Circuit)



**What it Does:**  
Packages any range of complicated electronics inside an easy to use package.

**Identifying:**  
The part ID is written on the outside of the package. (this sometimes requires a lot of light or a magnifying glass to read).

**No. of Leads:**  
2 - 100s (in this kit there is one with 3 (TMP36) and one with 16 (74HC595))

**Things to watch out for:**  
- Proper orientation. (look for marks showing pin 1)

**More Details:**  
<http://nedx.org/ICIC>

### Pushbutton



**What it Does:**  
Completes a circuit when it is pressed.

**Identifying:**  
A little square with leads out the bottom and a button on the top.

**No. of Leads:**  
4

**Things to watch out for:**  
- these are almost square so can be inserted 90 degrees off angle.

**More Details:**  
<http://nedx.org/BUTT>

### Potentiometer



**What it Does:**  
Produces a variable resistance dependant on the angular position of the shaft.

**Identifying:**  
They can be packaged in many different form factors, look for a dial to identify.

**No. of Leads:**  
3

**Things to watch out for:**  
- Accidentally buying logarithmic scale.

**More Details:**  
<http://nedx.org/POTE>

### Photo Resistor



**What it Does:**  
Produces a variable resistance dependant on the amount of incident light.

**Identifying:**  
Usually a little disk with a clear top and a curvy line underneath.

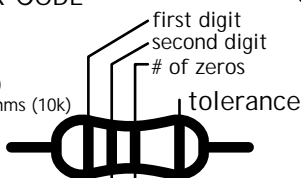
**No. of Leads:**  
2

**Things to watch out for:**  
- Remember it needs to be in a voltage divider before it provides a useful input.

**More Details:**  
<http://nedx.org/PHOT>

### RESISTOR COLOR CODE

Examples:  
green-blue-brown - 560 ohms  
red-red-red - 2 200 ohms (2.2k)  
brown-black-orange - 10 000 ohms (10k)



0 - Black	5 - Green	20% - none
1 - Brown	6 - Blue	10% - silver
2 - Red	7 - Purple	5% - gold
3 - Orange	8 - Grey	
4 - Yellow	9 - White	

### LEAD CLIPPING

Some components in this kit come with very long wire leads. To make them more compatible with a breadboard a couple of changes are required.

#### LEDs:

Clip the leads so the long lead is ~10mm (3/8") long and the short one is ~7mm (9/32").

#### Resistors:

Bend the leads down so they are 90 degrees to the cylinder. Then snip them so they are ~6mm (1/4") long.

#### Other Components:

Other components may need clipping. Use your discretion when doing so.

# 05 FIRS

first program

..:Your First:.

..:Program:..



## WHAT WE'RE DOING:

You have the development environment installed and your board unwrapped the next step is to write and upload your first program

..:A video walkthrough of this step can be viewed here:..  
..:http://nedx.org/TRBL:..

## THE STEPS:

### Step 1: Open The IDE

..:Start > Programs > Microsoft Visual Studio #### Express > Microsoft Visual C# #### Express:..

### Step 2: Register The IDE

..:Click "Obtain a registration key online":.. (Registration is free and only takes a couple of minutes)

..:Sign in using your .NET Passport, MSN Hotmail or MSN Messenger ID:.. (or create one)

..:Fill in a few details:.. (or scroll to the bottom and click OK)

..:Copy and paste the provided registration key:..

### Step 3: Start a New Project

..:Click "New Project":.. (top left)

..:Choose Visual C# > Micro Framework > Netduino Application:..

..:Name Your Application:.. (Type "FIRS" in the text-box at the bottom of the form)

..:OK:..

### Step 4: Write Your Code

..:Type the following into "Program.cs":.. (bottom of the right side menu)  
(save yourself some typing copy and paste the text from <http://nedx.org/FIRS> )

```
//Libraries that our program uses
using System; using System.Threading; using Microsoft.SPOT;
using Microsoft.SPOT.Hardware; using SecretLabs.NETMF.Hardware;
using SecretLabs.NETMF.Hardware.Netduino;
namespace FIRS // Define the namespace we are in ///
{
    public class Program
    {
        public static void Main() // The Main loop (run at power up) ///
        {
            OutputPort led = new OutputPort(Pins.ONBOARD_LED, false);
            // Define the Onboard LED as an output port
            while (true) // Do Forever ///
            {
                led.Write(true); //turn led on
                Thread.Sleep(100); //wait 250 milliseconds
                led.Write(false); //turn light off
                Thread.Sleep(100); //wait 250 milliseconds
            } // Close Forever Loop ///
        } // Close the Main() Loop ///
    } // Close the Program Loop ///
} // Close the Namespace Loop ///
```

### Step 5: Upload (plug in your Netduino board first)

..:Choose the Netduino Board:..

(top file menu) Project > FIRS Properties

(left side menu) .NET Micro Framework

(choose) Transport: USB & Device: Netduino\_Netduino

..: Upload Your Application:..

(top file menu) Debug > Start Debugging (F5)

..:Watch the Onboard LED Blink:..

(to return to editing the application) Debug > Stop Debugging (Shift+F5)

# 06 HWDD

hardware definitions

## VOLTAGE REGULATORS

Take a variable input voltage and provide the 5v and 3.3v the Netduino needs to run.

## USB CONNECTOR

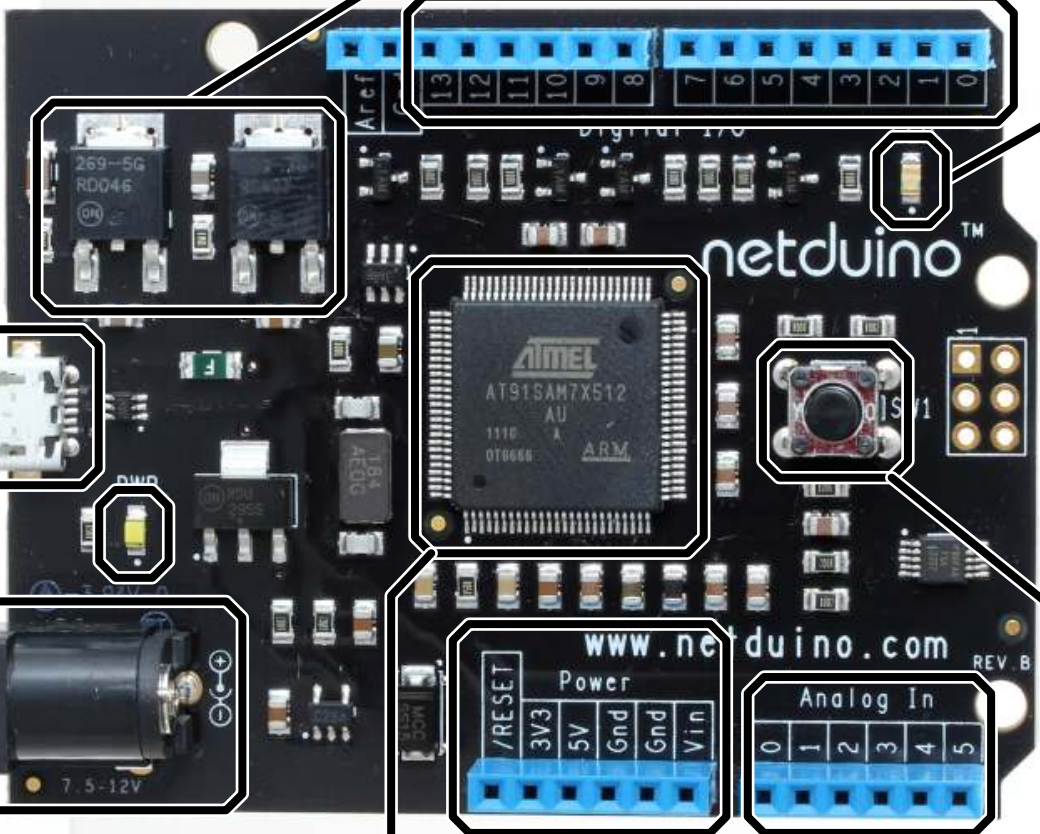
Plugs into your computer for either programming or debugging (micro USB)

## ONBOARD LED

Pi ns. ONBOARD\_LED  
An onboard LED that can be toggled in your programs

## DIGITAL PINS 0-13

Pi ns. GPI\_0\_PIN\_D0-D13  
Digital pins that can be configured as either inputs (reading a digital signal) or as outputs (being driven either high or low)



## PROCESSOR

The micro controller that your program runs on (Atmel AT91SAM, 32 bit 48 MHz)

## DC ADAPTER

Accepts between 7.5v and 12v. (2.1mm centre positive barrel jack)

## POWER PINS

Breakout pins for the 5v and 3.3v supplies, also ground and the Reset pin

## ANALOG INPUTS

Pi ns. GPI\_0\_PIN\_A0-A5  
Can be used to read a voltage between 0 and 3.3v. Useful for connecting sensors and potentiometers.

## RESET BUTTON

Pi ns. ONBOARD\_SW1  
Restarts the currently running program



### WHAT WE'RE DOING:

LEDs (light emitting diodes) are used in all sorts of clever things which is why we have included them in this kit. We will start off with something very simple, simply extending our getting started program to an externally wired LED. To get started, grab the parts listed below, pin the layout sheet to your breadboard and then plug everything in. Once the circuit is assembled you'll need to open a new Netduino application in Visual C# (like in the getting started program) and load the new program onto the Netduino (remember to switch the target from emulator to Netduino then press F5)

If you are having trouble a full video walkthrough of this program can be found here: <http://nedx.org/TRBL>

### THE CIRCUIT:

#### Parts:



NCIR-01  
Breadboard Sheet  
x1



2 Pin Header  
x4



10mm LED  
x1

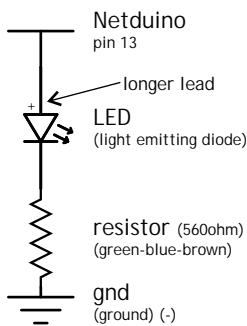


Wire



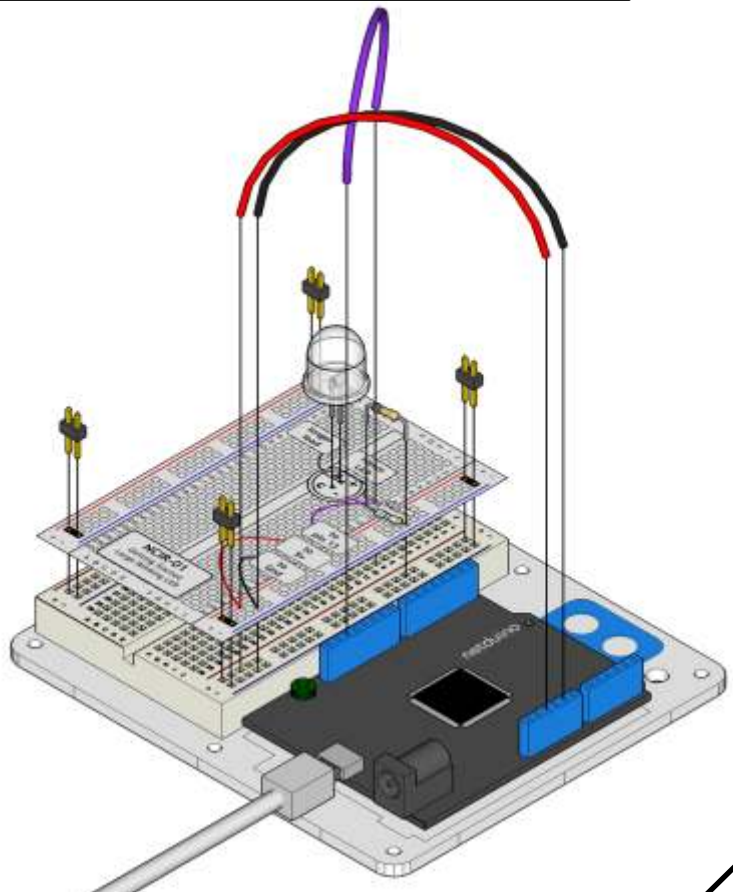
560 Ohm Resistor  
Green-Blue-Brown  
x1

#### Schematic



#### The Internet

..download:..  
breadboard layout sheet  
<http://nedx.org/NBLS01>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE01> )  
 (copy the text and paste it into an empty Netduino Project named NCIR01)

```
//Libraries that our program uses
using System; //use base classes
using System.Threading; //use threading classes
using Microsoft.SPOT; //use smart personal objects technology classes
using Microsoft.SPOT.Hardware; //use hardware related SPOT classes
using SecretLabs.NETMF.Hardware; //use Secret Labs hardware framework
using SecretLabs.NETMF.Hardware.Netduino; //use the Netduino specific classes

namespace NCIR01 // Define the namespace we are in ///
{
    public class Program
    {
        public static void Main() // The Main loop (run at power up) ///
        {
            // Setup Portion of Program, runs once at startup ///
            OutputPort led = new OutputPort(Pins.GPIO_PIN_D13, false); //define our LED as being connected to pin 13
            while (true) // Do Forever ///
            {
                led.Write(true); //turn led on
                Thread.Sleep(250); //wait 250 milliseconds
                led.Write(false); //turn light off
                Thread.Sleep(250); //wait 250 milliseconds
            } // Close Forever Loop ///
        } // Close the Main() Loop ///
    } // Close the Program Loop ///
} // Close the Namespace Loop ///
```

## NOT WORKING? (3 things to try)

### LED Not Lighting Up?

LEDs will only work in one direction. Try taking it out and twisting it 180 degrees. (no need to worry, installing it backwards does no permanent harm).

### Program Not Running?

If a black box appears when you start debugging, you need to switch from emulate mode to Netduino mode  
 Project>NCIR01 Properties>  
 .NET Micro Framework>  
 Transport: USB Device: Netduino

### Can't Change the Code?

To make changes to the code you must first stop Debugging.  
 Debug > Stop Debugging  
 (Shift F5)

## MAKING IT BETTER

### Changing the pin:

We've connected the LED to pin 13 but we can use any of the Netduino's digital pins. To change it take the wire plugged into pin 13 and move it to a pin of your choice (from 0-13)

### Then in the code change the line:

```
OutputPort led = new OutputPort(Pins.GPIO_PIN_D13, false)
OutputPort led = new OutputPort(Pins.GPIO_PIN_D--, false)
-- = new pin number
```

### Then debug the app: (F5)

### Change the blink time:

Unhappy with 1/4 second on 1/4 second off?

### In the code change the the two lines:

```
Thread.Sleep(250)
Thread.Sleep(---) --- = Delay in milliseconds
```

### Control the brightness:

Along with digital (on/off) control the Netduino can control some pins in an analog (brightness) fashion. (more details on this in later circuits). To play around with it.

### Change the LED to pin 9 and set it as a PWM output:

```
OutputPort led = new OutputPort(Pins.GPIO_PIN_D13, false);
PWM pwm = new PWM(Pins.GPIO_PIN_D9);
..also change the wire..
```

### Replace the code inside the { }'s of while (true) with this:

```
pwm.SetDutyCycle(new number);
```

(new number) = any number between 0 and 100.

0 = off, 100 = on, in between = different brightness

### Fading:

..Copy and Paste the code from <http://nedx.org/NCIRMB>..

Then start debugging (F5) to watch the LED fade in and then out.

### Making Even Better:

Trying playing around with some of the values. Don't worry it is very difficult to break anything.



### WHAT WE'RE DOING:

We have caused one LED to blink, now it's time to up the stakes. Lets connect eight. We'll also have an opportunity to stretch the Netduino a bit by creating various lighting sequences. This circuit is also a nice setup to experiment with writing your own programs and getting a feel for how the Netduino works.

Along with controlling the LEDs we start looking into a few simple programming methods to keep your programs small.

for() loops - used when you want to run a piece of code several times.  
arrays[] - used to make managing variables easier (it's a group of variables).

### THE CIRCUIT:

#### Parts:



NCIR-02  
Breadboard Sheet  
x1



2 Pin Header  
x4



5mm Green LED  
x8

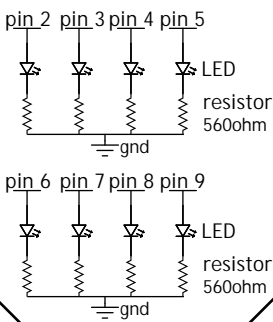


Wire



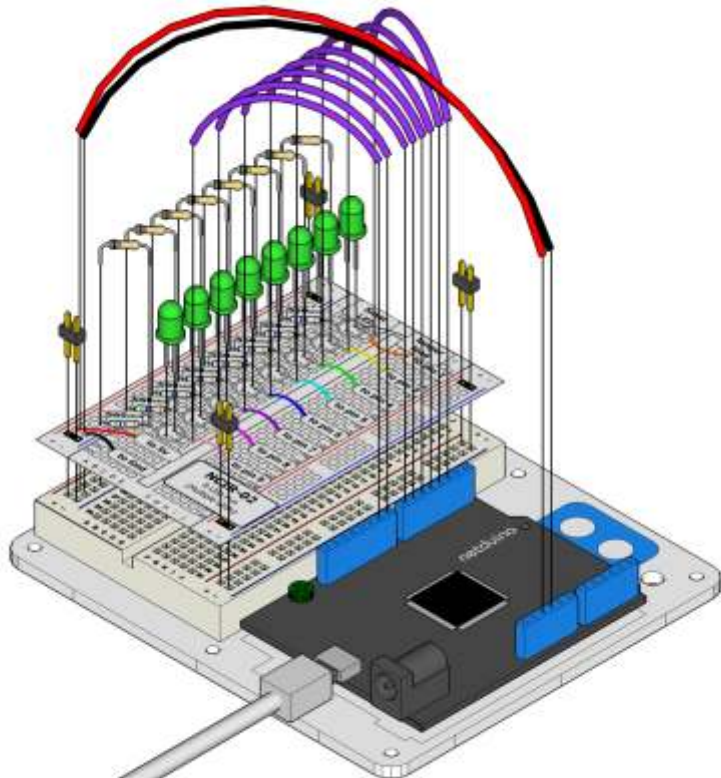
560 Ohm Resistor  
Green-Blue-Brown  
x8

#### Schematic



#### The Internet

download:  
breadboard layout sheet  
<http://nedx.org/NBLS02>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE02> )  
 (copy the text and paste it into an empty Netduino Project named NCIR02)

```

namespace NCIR02
{
    public class Program
    {
        static OutputPort[] leds = new OutputPort[8];
        //An array to hold the 8 pins the LEDs
        //are connected to

        public static void Main()
        {
            //Setup all 8 LED OutputPorts, one for
            //each connected pin
            leds[0] = new
                OutputPort(Pins.GPIO_PIN_D2, false);
            ...
            leds[7] = new
                OutputPort(Pins.GPIO_PIN_D9, false);

            while (true)          // Do Forever //
            {
                oneAfterAnotherNoLoop();
                //oneAfterAnotherLoop();
                //oneOnAtATime();
                //inAndOut();
            }

            // Will light one LED then delay for
            // delayTime then light the next LED until
            // all LEDs are on it will then turn them off
            // one after another.
            static void oneAfterAnotherNoLoop()
            {
                int delayTime = 100;

                leds[0].Write(true);
                Thread.Sleep(delayTime);

                leds[7].Write(true);
                Thread.Sleep(delayTime);

                leds[7].Write(false);
                Thread.Sleep(delayTime);

                leds[0].Write(false);
                Thread.Sleep(delayTime);
            }

            static void oneAfterAnotherLoop()
            {
                int delayTime = 100;

                //Turn each LED on one after another
                for (int i = 0; i <= 7; i++)
                {
                    leds[i].Write(true);
                    Thread.Sleep(delayTime);
                }

                for (int i = 7; i >= 0; i--)
                {
                    leds[i].Write(false);
                    Thread.Sleep(delayTime);
                }
            }
        }
    }
}
    
```

---more code in the download-able version---

## NOT WORKING? (3 things to try)

**Some LEDs Fail to Light**  
 It is easy to insert an LED backwards. Check the LEDs that aren't working and ensure they the right way around.

**Operating out of sequence**  
 With eight wires it's easy to cross a couple. Double check that the first LED is plugged into pin 2 and each pin there after.

**Starting Afresh**  
 Its easy to accidentally misplace a wire without noticing. Pulling everything out and starting with a fresh slate is often easier than trying to track down the problem.

## MAKING IT BETTER

Switching to loops:

Just beneath the `while (true)` statement there are 4 lines. The last three all start with a `'//'`. This means the line is treated as a comment (not run). To switch the program to use loops change these four lines to:

```

//oneAfterAnotherNoLoop();
oneAfterAnotherLoop();
//oneOnAtATime();
//inAndOut();
    
```

Upload the program, and notice that nothing has changed. You can take a look at the two functions, each does the same thing, but use different approaches (hint: the second one uses a for loop).

Extra animations:

Tired of this animation? Then try the other two sample animations. Un-comment their lines and upload the program to your board and enjoy the new light animations. (delete the slashes in front of row 3 and then 4)

Testing out your own animations:

Jump into the included code and start changing things. The main point is to turn an LED on use `leds[pinNumber].Write(true);` then to turn it off use `leds[pinNumber].Write(false);`. Type away, regardless of what you change you won't break anything.

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR02>



### WHAT WE'RE DOING:

The Netduino's pins are great for directly controlling small electric items like LEDs. However, when dealing with larger items (like a toy motor or washing machine), an external transistor is required. A transistor is incredibly useful. It switches a lot of current using a much smaller current. A transistor has 3 pins. For a negative type (NPN) transistor, you connect your load to the collector and the emitter to ground. Then when a small current flows from base to the emitter, a much larger current will flow through the transistor and your motor will spin (this happens when we set our Netduino pin HIGH (true)). There are literally thousands of different types of transistors, allowing every situation to be perfectly matched. We have chosen a P2N2222AG a rather common general purpose transistor. The important factors in our case are that its maximum voltage (40v) and its maximum current (600 milliamp) are both high enough for our toy motor (full details can be found on its datasheet <http://nedx.org/2222>).

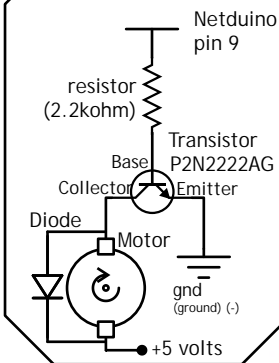
(The 1N4001 diode is acting as a flyback diode for details on why its there visit: <http://nedx.org/4001>)

### THE CIRCUIT:

#### Parts:

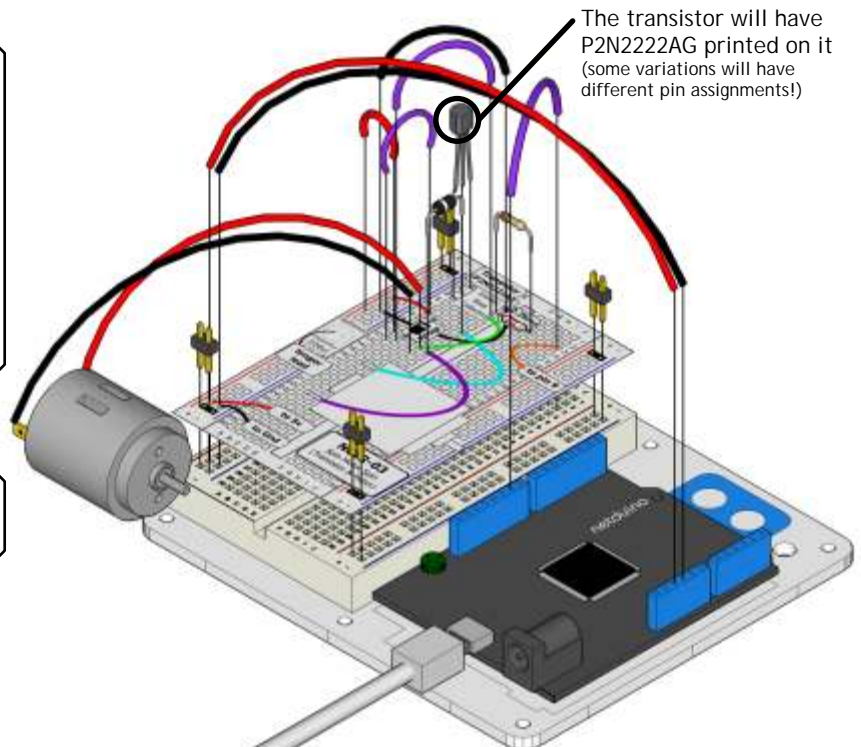
- |                             |                   |                                  |      |
|-----------------------------|-------------------|----------------------------------|------|
| NCIR-03 Breadboard Sheet x1 | 2 Pin Header x4   | Transistor P2N2222AG (TO92) x1   | Wire |
| Toy Motor x1                | Diode (1N4001) x1 | 2.2k Ohm Resistor Red-Red-Red x1 |      |

#### Schematic



#### The Internet

..download..  
breadboard layout sheet  
<http://nedx.org/NBLS03>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE03> )  
(copy the text and paste it into an empty Netduino Project named NCIR03)

```
namespace NCIR03
{
    public class Program
    {
        public static void Main()
        {
            while (true)
            {
                motorOnThenOff();
                //motorOnThenOffWithSpeed();
                //motorAcceleration();
            }
        }
    }
}

/// motorOnThenOff() - turns motor on then off
/// (notice this code is identical to the code we
/// used for the blinking LED)
static void motorOnThenOff()
{
    OutputPort motor = new
        OutputPort(Pins.GPIO_PIN_D9, false);

    int onTime = 2500; //the number of
        // milliseconds for the motor to turn on for
    int offTime = 1000; //the number of
        // milliseconds for the motor to turn off for

    motor.Write(true); // turns the motor On
    Thread.Sleep(onTime); // waits for onTime milliseconds
    motor.Write(false); // turns the motor Off

    Thread.Sleep(offTime); // waits for offTime milliseconds
    motor.Dispose();
}

// motorOnThenOffWithSpeed() - turns motor on
// then off but uses speed values as well
static void motorOnThenOffWithSpeed()
{
    PWM motor = new PWM(Pins.GPIO_PIN_D9);
    uint onSpeed = 100; // a number between 0
        // (stopped) and 100 (full speed)
    int onTime = 2500; //the number of
        // milliseconds for the motor to turn on for
    uint offSpeed = 50; // a number between 0
        // (stopped) and 100 (full speed)
    int offTime = 1000; //the number of
        // milliseconds for the motor to turn off for
    motor.SetDutyCycle(onSpeed);
        // turns the motor On
    Thread.Sleep(onTime);
        // waits for onTime milliseconds
    motor.SetDutyCycle(offSpeed);
        // turns the motor Off
    Thread.Sleep(offTime);
        // waits for offTime milliseconds
    motor.Dispose();
}

---more code in the download-able version---
```

## NOT WORKING? (3 things to try)

### Motor Not Spinning?

If you sourced your own transistor, double check with the data sheet that the pinout is compatible with a P2N2222A (many are reversed).

### Still No Luck?

If you sourced your own motor, double check that it will work with 5 volts and that it does not draw too much power.

### Still Not Working?

Sometimes the Netduino board will disconnect from the computer. Try un-plugging and then re-plugging it into your USB port.

## MAKING IT BETTER

### Controlling speed:

We played with the Netduino's ability to control the brightness of an LED earlier now we will use the same feature to control the speed of our motor. The Netduino does this using something called Pulse Width Modulation (PWM). This relies on the Netduino's ability to operate really, really fast. Rather than directly controlling the voltage coming from the pin the Netduino will switch the pin on and off very quickly. In the computer world this is going from 0 to 3.3 volts many times a second, but in the human world we see it as a voltage. For example if the Netduino is PWM'ing at 50% we see the light dimmed 50% because our eyes are not quick enough to see it flashing on and off. The same feature works with transistors. Don't believe me? Try it out.

### Change the code after the while (true) line to:

```
// motorOnThenOff();
    motorOnThenOffWithSpeed();
// motorAcceleration();
Then debug the application. You can change the speeds by changing the variables onSpeed and offSpeed.
```

### Accelerating and decelerating:

Why stop at two speeds, why not accelerate and decelerate the motor. To do this simply change the code after the while (true) line to:

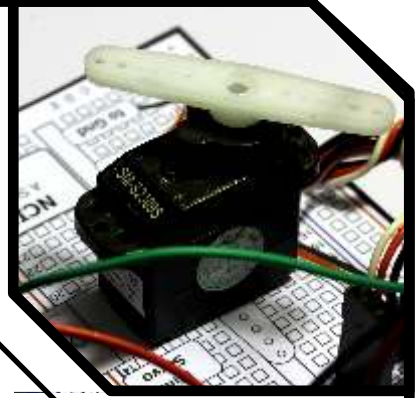
```
// motorOnThenOff();
// motorOnThenOffWithSpeed();
    motorAcceleration();
```

Then debug the application and watch as your motor slowly accelerates up to full speed then slows down again. If you would like to change the speed of acceleration change the variable delayTime (larger means a longer acceleration time).

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR03>



## WHAT WE'RE DOING:

Spinning a motor is good fun but when it comes to projects where motion control is required they tend to leave us wanting more. The answer? Hobby servos. They are mass produced, widely available and cost anything from a couple of dollars to hundreds. Inside is a small gearbox (to make the movement more powerful) and some electronics (to make it easier to control). A standard servo is position-able from 0 to 180 degrees. Positioning is controlled through a timed pulse, between 1.25 milliseconds (0 degrees) and 1.75 milliseconds (180 degrees) (1.5 milliseconds for 90 degrees). Timing varies between manufacturer. If the pulse is sent every 25-50 milliseconds the servo will run smoothly. To achieve this on the Netduino we will use its PWM function.

## THE CIRCUIT:

### Parts:



NCIR-04 Breadboard Sheet x1



2 Pin Header x4



3 Pin Header x1

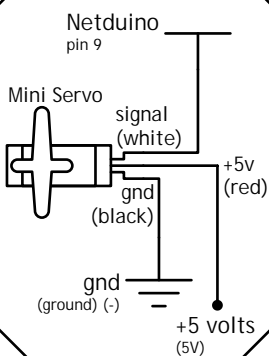


Wire



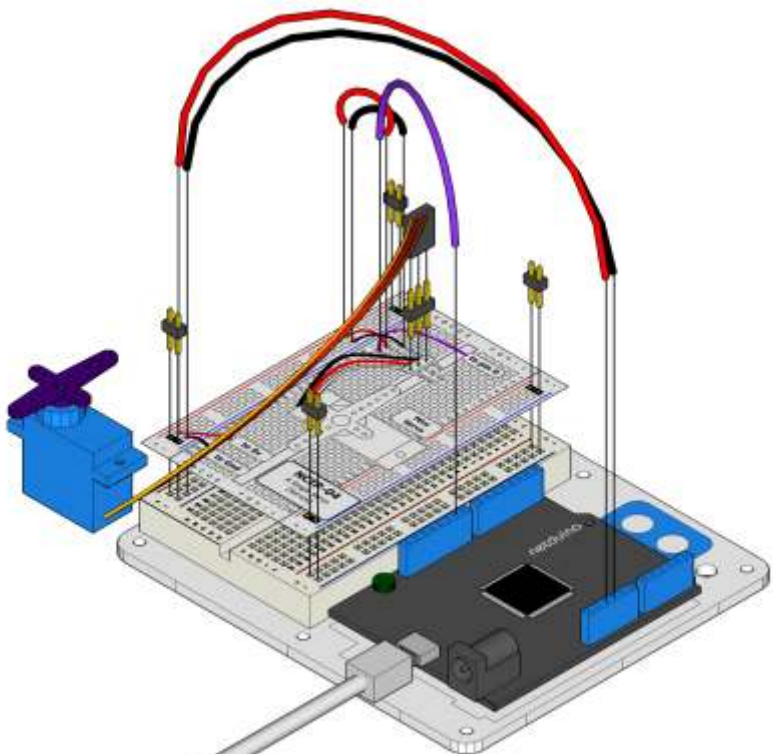
Mini Servo x1

### Schematic



### The Internet

..:download:..  
breadboard layout sheet  
<http://nedx.org/NBLS04>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE04> )

(copy the text and paste it into an empty Netduino Project named NCIR04)

```
namespace NCIR04
{
    public class Program
    {
        public static void Main()
        {
            PWM servo =
                new PWM(Pi ns. GPIO_PIN_D9);
            //creates a new PWM object the
            //servo is connected to
            while (true) /// Do Forever ///
            {
                for (int pos = 0; pos < 180; pos++)
                {
                    setServo(servo, pos);
                    // tell servo to go to
                    //position in variable 'pos'
                }
                for (int pos = 180; pos > 0; pos--)
                {
                    setServo(servo, pos);
                    // tell servo to go to
                    //position in variable 'pos'
                }
            }
        }
    }
}

/// Sets a servo to the desired angle
public static void setServo
(PWM servo, int position)
{
    if (position > 180) position = 180;
    if (position < 0) position = 0;

    int min = 500;
    int max = 2500;
    int range = max - min;

    float pulse = (((float)max -
        (((float)180 - (float)position) /
        (float)180) * (float)range));

    servo.SetPulse(20000, (uint)pulse);
}
```

## NOT WORKING? (3 things to try)

### Servo Not Twisting?

Even with colored wires it is still shockingly easy to plug a servo in backwards. This might be the case.

### Still Not Working

A mistake we made a time or two was simply forgetting to connect the power (red and black wires) to +5 volts and ground.

### Fits and Starts

If the servo begins moving then twitches, and there's a flashing light on your Netduino board, the power supply you are using is not quite up to the challenge. Using a fresh battery instead of USB should solve this problem.

## MAKING IT BETTER

### Changing the Sweep Speed:

Not happy with the speed the servo is moving? You can change this by altering the code. Simply change:

```
for (int pos = 0; pos < 180; pos++)
for (int pos = 0; pos < 180; pos = pos + #)
and
for (int pos = 0; pos > 180; pos--)
for (int pos = 0; pos > 180; pos = pos - #)
```

# = degrees per step.

### Send to a Specific Angle:

To get the servo to go to a specific angle rather than sweep, change the code after `while (true)` to:

```
setServo(servo, ###);
```

### = angle between 0 & 180 degrees

### Mixing and Matching:

Each circuit in this guide is rather basic. The real fun comes when you start using multiple components and real world items to make something neat. The controlled movement of a servo is very helpful in this, so get thinking and creating.

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR04>



### WHAT WE'RE DOING:

Time to start playing with chips, or integrated circuits (ICs) as they like to be called. The external packaging of a chip can be very deceptive. For example, the chip on the Netduino board (a micro controller) and the one we will use in this circuit (a shift register) look slightly similar but are in fact rather different. The price of the Atmel chip on the Netduino board is a few dollars while the 74HC595 is a couple dozen cents. It's a good introductory chip, and once you're comfortable playing around with it and its datasheet (available online <http://nedx.org/74HC595>) the world of chips will be your oyster. The shift register (also called a serial to parallel converter), will give you an additional 8 outputs (to control LEDs and the like) using only three Netduino pins. They can also be linked together to give you a nearly unlimited number of outputs using the same three pins. To use it you "clock in" the data and then lock it in (latch it). To do this you set the data pin to either HIGH or LOW, pulse the clock, then set the data pin again and pulse the clock repeating until you have shifted out 8 bits of data. Then you pulse the latch and the 8 bits are transferred to the shift registers pins. It sounds complicated but is really simple once you get the hang of it.

(for a more in depth look at how a shift register works visit: <http://nedx.org/SHIF>)


### THE CIRCUIT:

#### Parts:

- 

NCIR-05  
Breadboard Sheet  
x1
- 

2 Pin Header  
x4
- 

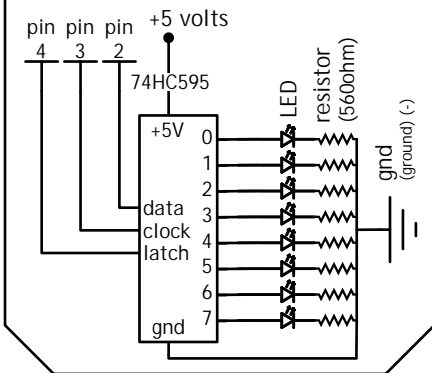
Shift Register  
74HC595  
x1
- 

Wire
- 

Red LED  
x8
- 

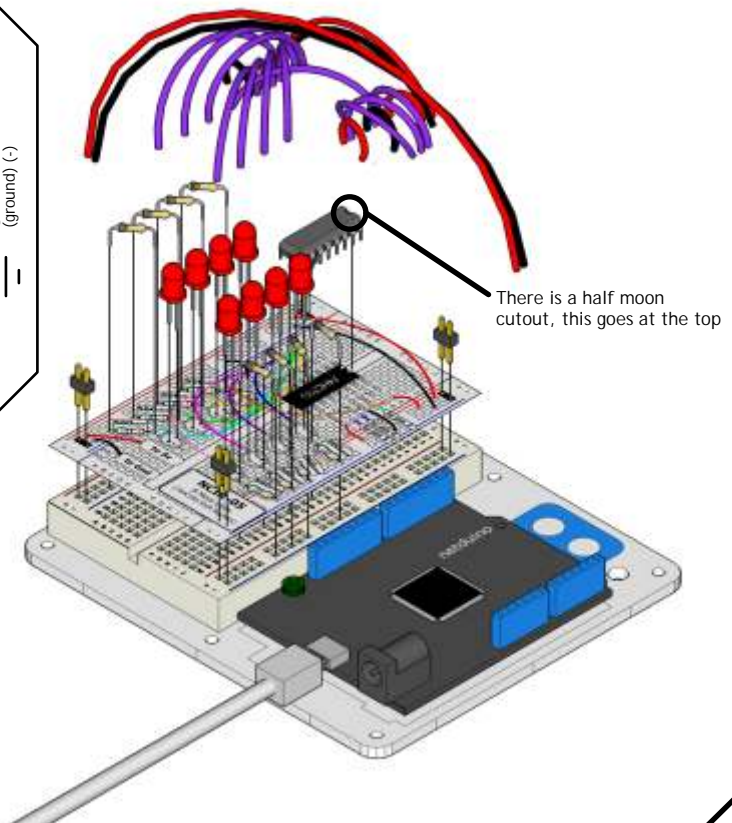
560 Ohm Resistor  
Green-Blue-Brown  
x8

#### Schematic



#### The Internet

download:  
breadboard layout sheet  
<http://nedx.org/NBLS05>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE05> )  
 (copy the text and paste it into an empty Netduino Project named NCIR05)

```
namespace NCIR05
{
    public class Program
    {
        static OutputPort data = new
            OutputPort(Pins.GPIO_PIN_D2, false);
        static OutputPort clock = new
            OutputPort(Pins.GPIO_PIN_D3, false);
        static OutputPort latch = new
            OutputPort(Pins.GPIO_PIN_D4, false);

        public static void Main()
        {
            int delayTime = 100;
            while (true) // Do Forever ///
            {
                for (int i = 0; i < 256; i++)
                {
                    updateLEDs(i);
                    Thread.Sleep(delayTime);
                }
            }

            /// sends the LED states set in value to the
            /// 74HC595 sequence
            static void updateLEDs(int value)
            {
                latch.Write(false); //Pulls the chips latch low
                for (int i = 0; i < 8; i++)
                {
                    int bit = value & 0x80;
                    value = value << 1;

                    if (bit == 128)
                    {
                        data.Write(true);
                    }
                    else
                    {
                        data.Write(false);
                    }

                    clock.Write(true);
                    Thread.Sleep(1);
                    clock.Write(false);
                }
                latch.Write(true);
            }

            //These are used in the bitwise math that we use to
            //change individual LEDs
            static int[] bits = {1, 2, 4, 8, 16, 32, 64, 128};
            static int[] masks = {254, 253, 251, 247, 239, 223,
                191, 127};
            static int ledState = 0;

            /// Changes an individual LED
            public static void changeLED(int led, bool state){
                ledState = ledState & masks[led];
                if (state){ledState = ledState | bits[led];}
                updateLEDs(ledState);
            }
        }
    }
}
```

## NOT WORKING? (3 things to try)

### The Netduino's power LED goes out

This happened to us a couple of times, it happens when the chip is inserted backwards. If you fix it quickly nothing will break.

**Not Quite Working**  
 Sorry to sound like a broken record but it is probably something as simple as a crossed wire.

**Frustration?**  
 Don't worry the Netduino has a great community around it. There's always someone keen to help out on the forums. <http://nedx.org/FORU>

## MAKING IT BETTER

**Understanding What's Going On:**  
 The Netduino makes rather complex actions easy to hide, shifting out data is one of these cases. Take a look at the updateLEDs(int value). If you look at the code you can see how we are communicating with the chip one bit at a time. (for more details <http://nedx.org/SPI>).

**Controlling individual LEDs:**  
 Time to start controlling the LEDs in a similar method as we did in NCIR02. As the eight LED states are stored in one byte (an 8 bit value) for details on how this works visit <http://nedx.org/BINA>. A Netduino is very good at manipulating bits and there are an entire set of operators that help us out. Details on bitwise maths (<http://nedx.org/BI TW>).

**Our implementation.**  
 Replace the code after while (true) with:

```
for (int i = 0; i < 8; i++)
{
    changeLED(i, true);
    Thread.Sleep(delayTime);
}
for (int i = 0; i < 8; i++)
```

```
{
    changeLED(i, false);
    Thread.Sleep(delayTime);
}
```

Uploading this will cause the lights to light up one after another and then off in a similar manner to NCIR02. Check the code and wikipedia to see how it works.

**More animations:**  
 Now things get more interesting. If you look back to the code from NCIR02 (8 LED Fun) you see we change the LEDs using leds[ # ]. Write(true), this is a similar format as the routine we wrote changeLED(led#, state). You can use the animations you wrote for NCIR02 by copying the code into this sketch and changing all the leds[ # ]. Write(true)'s to changeLED()'s. Powerful? Very. (you'll also need to change a few other things but follow the errors and it works itself out).

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR05>



### WHAT WE'RE DOING:

To this point we have controlled light, motion, and electrons. Let's tackle sound next. But sound is an analog phenomena, how will our digital Netduino cope? We will once again rely on its incredible speed which will let it mimic analog behavior. To do this, we will attach a piezo element to one of the Netduino's digital pins. A piezo element makes a clicking sound each time it is pulsed with current. If we pulse it at the right frequency (for example 440 times a second to make the note middle A) these clicks will run together to produce notes. Let's get to experimenting with it and get your Netduino playing "Twinkle Twinkle Little Star".

### THE CIRCUIT:

#### Parts:



NCIR-06  
Breadboard Sheet  
x1



2 Pin Header  
x4

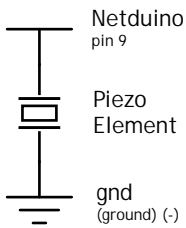


Piezo Element  
x1



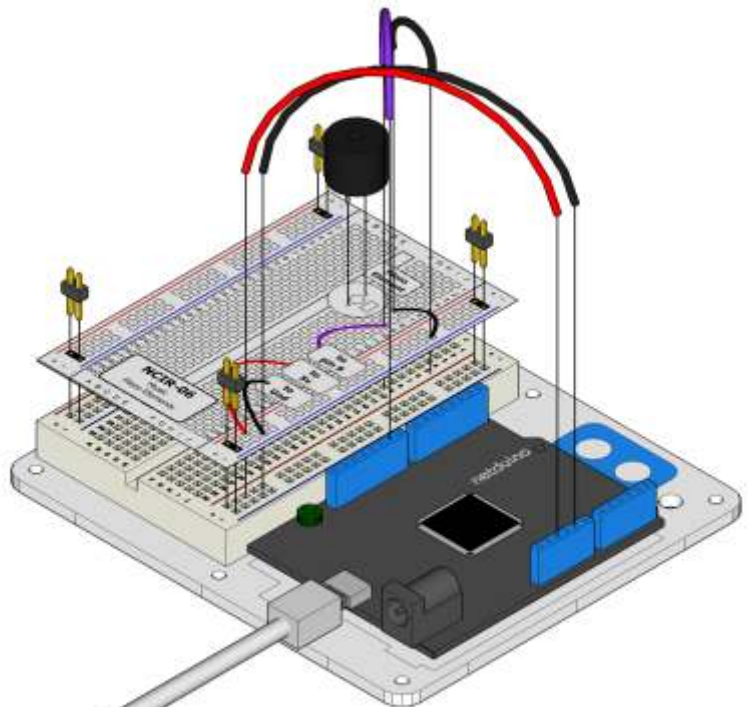
Wire

#### Schematic



#### The Internet

..download:  
breadboard layout sheet  
<http://nedx.org/NBLS06>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE06> )  
 (copy the text and paste it into an empty Netduino Project named NCIR06)

```
namespace NCIR06
{
    public class Program
    {
        static PWM speaker = new PWM(Pins.GPIO_PIN_D9);
        static char[] notes =
            "ccggaagffeeddc ".ToCharArray();
        static int[] beats =
            {1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 4 };
        static int tempo = 300;

        public static void Main()
        {
            while (true)          /// Do Forever ///
            {
                for (int i = 0; i < notes.Length; i++)
                {
                    playNote(notes[i], beats[i] * tempo);
                    playNote(' ', tempo / 2);
                }
            }

            /// Plays a particular tone for a particular
            /// duration
            static void playTone(int tone, int duration)
            {
                speaker.SetPulse((uint)(tone * 2), (uint)tone);
                Thread.Sleep(duration);
            }

            /// Looks up the tone for a note based on it's
            /// letter and plays it for a specific amount of
            static void playNote(char note, int duration)
            {
                char[] names =
                    {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C', ' ' };
                int[] tones =
                    {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956, 0 };
                for (int i = 0; i < 9; i++)
                {
                    if (names[i] == note)
                    {
                        playTone(tones[i], duration);
                    }
                }
            }
        }
    }
}
```

## NOT WORKING? (3 things to try)

### No Sound

Given the size and shape of the piezo element it is easy to miss the right holes on the breadboard. Try double checking its placement.

### Can't Think While the Melody is Playing?

Just pull up the piezo element whilst you think, debug your application then plug it back in.

### Tired of Twinkle Twinkle Little Star?

The code is written so you can easily add your own songs, check out the code below to get started.

## MAKING IT BETTER

Playing with the speed:

The timing for each note is calculated based on variables, as such we can tweak the sound of each note or the timing. To change the speed of the melody you need to change only one line.

```
static int tempo = 300;
```

```
static int tempo = (new #)
```

Change it to a larger number to slow the melody down, or a smaller number to speed it up.

Tuning the notes:

If you are worried about the notes being a little out of tune this can be fixed as well. The notes have been calculated based on a formula in the comment block at the top of the program. But to tune individual notes just adjust their values in the tones[] array up or down until they sound right. (each note is matched by its name in the names[] (array ie. c = 1915 )

```
char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C', ' ' };
int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956, 0};
```

Composing your own melodies:

The program is pre-set to play "Twinkle Twinkle Little Star" however the way it is programmed makes changing the song easy. Each song is defined in two arrays, the first array notes[] defines each note, and the second beats[] defines how long each note is played. Some Examples:

### Twinkle Twinkle Little Star

```
static char[] notes =
    "ccggaagffeeddc ".ToCharArray();
static int[] beats =
    {1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 4};
```

### Happy Birthday (first line)

```
static char[] notes =
    "ccdcFecdcgff ".ToCharArray();
static int[] beats =
    {1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 4};
```

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR06>



### WHAT WE'RE DOING:

Up to this point we have focused entirely on outputs, time to get our Netduino to listen, watch and feel. We'll start with a simple pushbutton. Wiring up the pushbutton is easy. There is one component, the pull up resistor, that might seem out of place. This is included because a Netduino doesn't sense the same way we do (ie button pressed, button unpressed). Instead it looks at the voltage on the pin and decides whether it is HIGH or LOW. The button is set up to pull the Netduino's pin LOW when it is pressed, however, when the button is unpressed the voltage of the pin will float (causing occasional errors). To get the Netduino to reliably read the pin as HIGH when the button is unpressed, we add the pull up resistor.

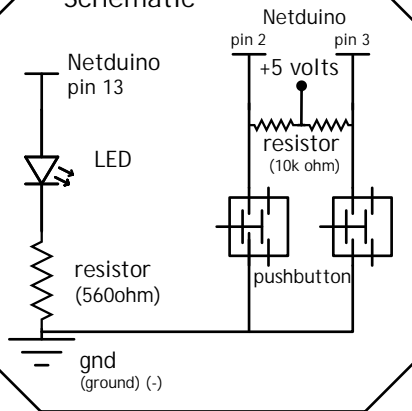
(note: the first example program uses only one of the two buttons)

### THE CIRCUIT:

#### Parts:

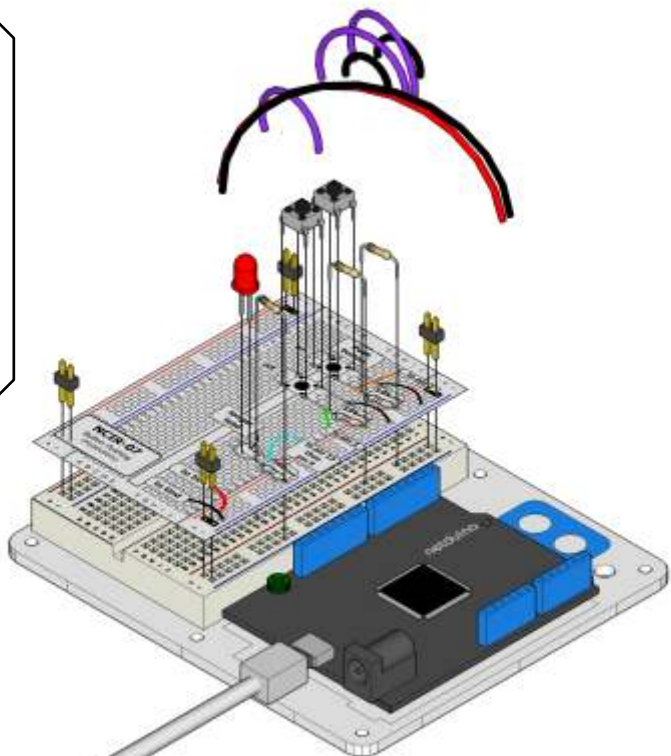
- NCIR-07 Breadboard Sheet x1
- 2 Pin Header x4
- Pushbutton x2
- Wire
- 10k Ohm Resistor Brown-Black-Orange x2
- 560 Ohm Resistor Green-Blue-Brown x1
- Red LED x1

#### Schematic



#### The Internet

download:  
breadboard layout sheet  
<http://nedx.org/NBLS07>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE07> )  
 (copy the text and paste it into an empty Netduino Project named NCIR07)

```

namespace NCIR07
{
    public class Program
    {
        static InputPort button1 = new
            InputPort(Pins.GPIO_PIN_D2, false,
                Port.ResistorMode.Disabled);
        //define pin 2 as an input for button #1
        static InputPort button2 = new
            InputPort(Pins.GPIO_PIN_D3, false,
                Port.ResistorMode.Disabled);
        //define pin 3 as an input for button #2
        static OutputPort led = new
            OutputPort(Pins.GPIO_PIN_D13, false);
        //define our LED as being connected to
        //pin 13

        public static void Main()
        {
            while (true) // Do Forever ///
            {
                turnLEDOnWithButton();
                //Turns the LED on when button one is
                //pressed and off when not
                //turnLEDOnOffWithButtons();
                //Turns the LED on when button 1 is
                //pressed and off when button 2 is
                //pressed
            }
        }
    }
}

// If button 1 is pressed the light is turned on
// If it isn't the light is turned off
static void turnLEDOnWithButton()
{
    if (button1.Read())
        //Checks if the button is pressed
    {
        led.Write(false);
        //If button 1 isn't pressed turn the light off
    }
    else
    {
        led.Write(true);
        //If button 1 is pressed turn the light on
    }
}

// If button 1 is pressed the light is turned on.
// If button 2 is pressed the light is turned off.
static void turnLEDOnOffWithButtons()
{
    if (button1.Read() == false) led.Write(true);
    //If button 1 is pressed turn the LED on
    if (button2.Read() == false) led.Write(false);
    //If button 2 is pressed turn the LED off
}

```

## NOT WORKING? (3 things to try)

**Light Not Turning On**  
 The pushbutton is square and because of this it is easy to put it in the wrong way. Give it a 90 degree twist and see if it starts working.

**Light Not Fading**  
 A bit of a silly mistake we constantly made, when you switch from simple on off to fading remember to move the LED wire from pin 13 to pin 9.

**Underwhelmed?**  
 No worries these circuits are all super stripped down to make playing with the components easy, but once you throw them together the sky is the limit.

## MAKING IT BETTER

On button off button:  
 The initial example may be a little underwhelming (ie. I don't really need a Netduino to do this), let's make it a little more complicated. One button will turn the LED on the other will turn the LED off. Change the code after while (true) to:

```

//turnLEDOnWithButton();
turnLEDOnOffWithButtons();

```

Debug the program to your board, and start toggling the LED on and off.

Fading up and down:  
 Lets use the buttons to control an analog signal. To do this you will need to change the wire connecting the LED from pin 13 to pin 9.

Next copy and paste the code from:  
<http://nedx.org/CODE07MB>  
 Into a blank Netduino project

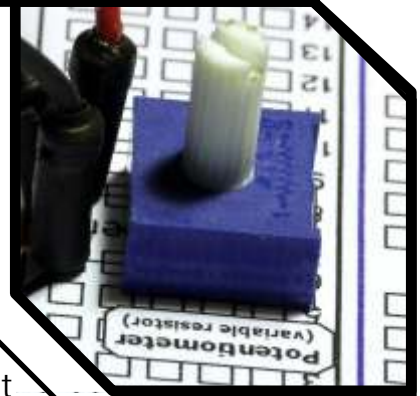
Now when you press button 1 the light will increase in brightness, and when you press button 2 it will decrease. To see how this is done, look at the comments included with the code.

Changing fade speed:  
 If you would like the LED to fade faster or slower, there is only one line of code that needs changing;  
 Thread.Sleep(10); ----> Thread.Sleep(new #);  
 To fade faster make the number smaller, slower requires a larger number.

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR07>



### WHAT WE'RE DOING:

Along with the digital pins, the Netduino also has 6 pins which can be used for analog input. These inputs take a voltage (from 0 to 3.3 volts) and convert it to a digital number between 0 (0 volts) and 1023 (3.3 volts) (10 bits of resolution). A very useful device that exploits these inputs is a potentiometer (also called a variable resistor). When it is connected with 3.3 volts across its outer pins the middle pin will read some value between 0 and 3.3 volts dependent on the angle to which it is turned (ie. 1.67 volts in the middle). We can then use the returned values as a variable in our program.

### THE CIRCUIT:

#### Parts:



NCIR-08 Breadboard Sheet x1



2 Pin Header x4



Potentiometer 10k ohm x1



Wire

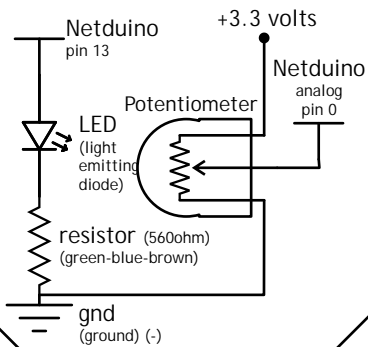


Green LED x1



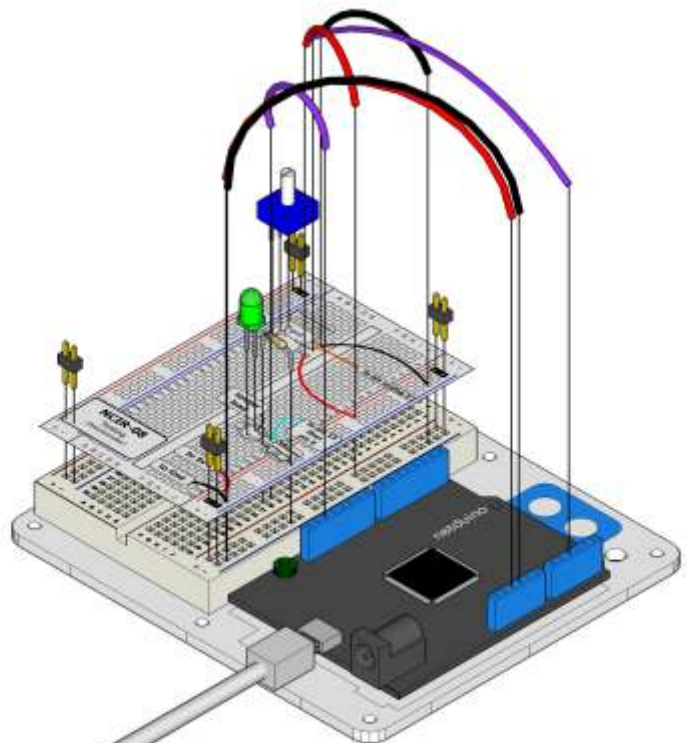
560 Ohm Resistor Green-Blue-Brown x1

#### Schematic



#### The Internet

..download:.  
breadboard layout sheet  
<http://nedx.org/NBLS08>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE08> )  
 (copy the text and paste it into an empty Netduino Project named NCIR08)

```
namespace NCIR08
public class Program
{
  static OutputPort led = new
  OutputPort(Pins.GPIO_PIN_D13,
  false);
  // defines pin 13 as
  // connected to our LED
  static AnalogInput potentiometer =
  new AnalogInput(Pins.GPIO_PIN_A0);
  // defines our potentiometer
  // as being connected to
  // analog pin 0
  static int sensorValue = 0;
  // variable to store our
  // sensor value

  public static void Main()
  {
    while (true)///Do Forever///
    {
      sensorValue = potentiometer.Read();
      // Read the potentiometer's value
      led.Write(true); // Turns the LED on
      Thread.Sleep(sensorValue);
      // Waits sensorValue
      // milliseconds
      led.Write(false); // Turns the LED off
      Thread.Sleep(sensorValue);
      // Waits sensorValue
      // milliseconds
    }
  }
}
```

## NOT WORKING? (3 things to try)

### Sporadically Working

This is most likely due to a slightly dodgy connection with the potentiometer's pins. This can usually be conquered by taping the potentiometer down.

### Not Working

Make sure you haven't accidentally connected the potentiometer's wiper to digital pin 0 rather than analog pin 0. (the row of pins beneath the power pins)

### Still Backward

You can try operating the circuit upside down. Sometimes this helps.

## MAKING IT BETTER

### Threshold switching:

Sometimes you will want to switch an output when a value exceeds a certain threshold. To do this with a potentiometer change the code to after `while (true)` to:

```
sensorValue = potentiometer.Read();
int threshold = 512;
if (sensorValue > threshold)
{
  led.Write(true); // Turn the LED on
}
else
{
  led.Write(false); // Turn the LED off
}
```

This will cause the LED to turn on when the value is above 512 (about halfway), you can adjust the sensitivity by changing the threshold value.

### Fading:

Let's control the brightness of an LED directly from the potentiometer. To do this we need to first change the pin the LED

is connected to. Move the wire from pin 13 to pin 9

Then change the output pin declaration from:  

```
static OutputPort led = new
OutputPort(Pins.GPIO_PIN_D13, false);
static PWM led = new PWM(Pins.GPIO_PIN_D9);
```

Then change the code after `while (true)` to:

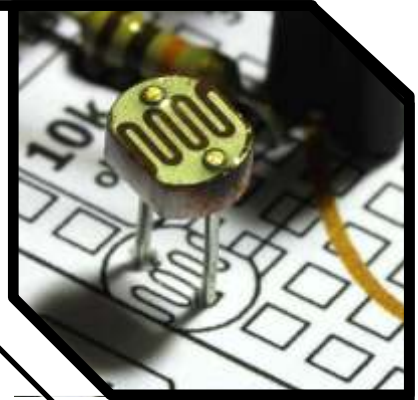
```
sensorValue = potentiometer.Read();
sensorValue = (int)((double)sensorValue * 0.0977);
led.SetDutyCycle((uint)sensorValue);
```

Upload the code and watch as your LED fades in relation to your potentiometer spinning. (Note: the reason we multiply sensorValue by 0.0977 is the .read() function returns a value from 0 to 1023 (10 bits), and .setDutyCycle() takes a value from 0 to 100.

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR08>





### WHAT WE'RE DOING:

Whilst getting input from a potentiometer can be useful for human controlled experiments, what do we use when we want an environmentally controlled experiment? We use exactly the same principles but instead of a potentiometer (twist based resistance) we use a photo resistor (light based resistance). The Netduino cannot directly sense resistance (it senses voltage) so we set up a voltage divider (<http://nedx.org/VODI>). The exact voltage at the sensing pin is calculable, but for our purposes (just sensing relative light) we can experiment with the values and see what works for us. A low value will occur when the sensor is well lit while a high value will occur when it is in darkness.

### THE CIRCUIT:

#### Parts:

- 

NCIR-09  
Breadboard Sheet  
x1
- 






2 Pin Header  
x4
- 

Photo-Resistor  
x1
- 

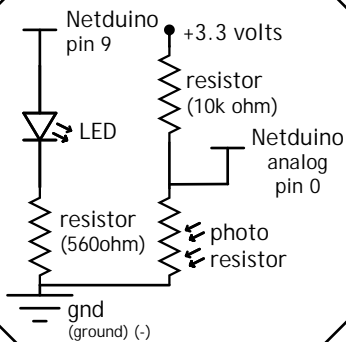
Wire
- 

10k Ohm Resistor  
Brown-Black-Orange  
x1
- 

560 Ohm Resistor  
Green-Blue-Brown  
x1
- 

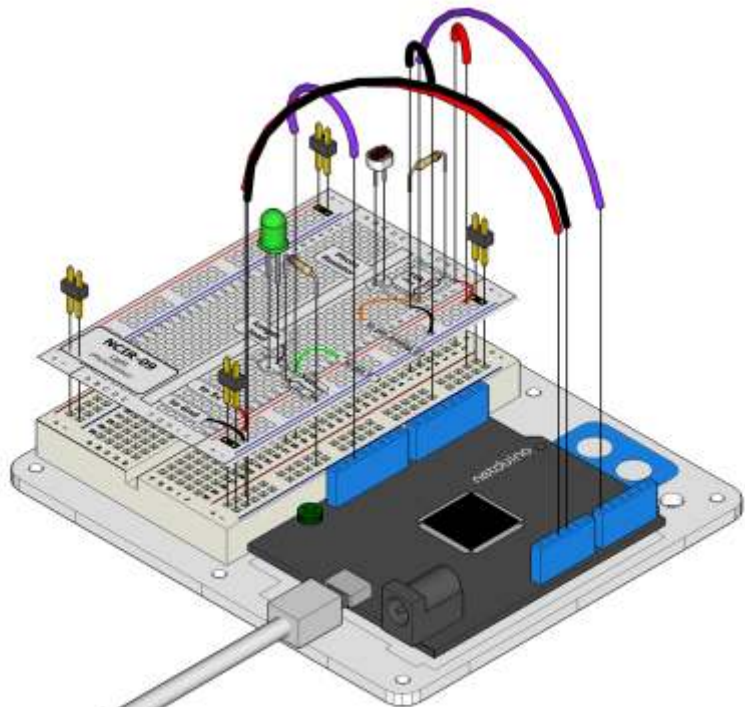
Green LED  
x1

#### Schematic



#### The Internet

..download:..  
breadboard layout sheet  
<http://nedx.org/NBLS09>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE09> )  
 (copy the text and paste it into an empty Netduino Project named NCIR09)

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware;
using SecretLabs.NETMF.Hardware.Netduino;

namespace NCIR09
{
    public class Program
    {
        static AnalogInput lightSensor = new
        AnalogInput(Pins.GPIO_PIN_A0);
        static OutputPort led = new
        OutputPort(Pins.GPIO_PIN_D9, false);

        public static void Main()
        {
            int value = 0;
            while (true)
            {
                value = lightSensor.Read();
                if (value > 300)
                {
                    led.Write(true);
                }
                else
                {
                    led.Write(false);
                }
            }
        }
    }
}

Debug.Print(value.ToString());
Thread.Sleep(100);
```

## NOT WORKING? (3 things to try)

### LED Remains Dark

This is a mistake we continue to make time and time again, if only they could make an LED that worked both ways. Pull it up and give it a twist.

### It Isn't Responding to Changes in Light.

Given that the spacing of the wires on the photo-resistor is not standard, it is easy to misplace it. Double check its in the right place.

### Still not quite working?

You may be in a room which is either too bright or dark. Try turning the lights on or off to see if this helps. Or if you have a flashlight near by give that a try.

## MAKING IT BETTER

Reverse the response:  
 Perhaps you would like the opposite response. Don't worry we can easily reverse this just change:

```
led.SetDutyCycle((uint)val ue);
```

```
led.SetDutyCycle(100 - (ui nt)val ue);
```

Upload and watch the change:

### Night light:

Rather than controlling the brightness of the LED in response to light, let's instead turn it on or off based on a threshold value. Replace the code after while (true) with.

```
int threshold = 300;
val ue = lightSensor.Read();
if (val ue > threshold)
{
    led.SetDutyCycle(100);
}
```

```
}
else
{
    led.SetDutyCycle(0);
}
```

To change the point where the LED turns on or off change the threshold value.

For this example we use the PWM function on pin 9. As we are just turning the LED on or off, you could switch to using an OutputPort instead (this would allow you to use any pin rather than just those compatible with PWM).

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR09>

# NCIR-10

.:Temperature:.

.:TMP36 Temperature Sensor:.



## WHAT WE'RE DOING:

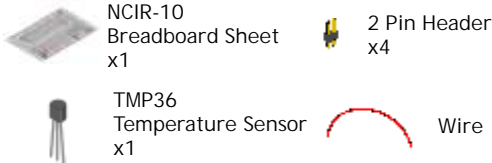
What's the next phenomena we will measure with our Netduino? Temperature. To do this we'll use a rather complicated IC (integrated circuit) hidden in a package identical to our P2N2222AG transistors. It has three pin's, ground, signal and +3.3 volts, and is easy to use. It outputs 10 millivolts per degree centigrade on the signal pin (to allow measuring temperatures below freezing there is a 500 mV offset eg. 25° C = 750 mV, 0° C = 500mV). To convert this from the digital value to degrees, we will use some of the Netduino's maths abilities. Then to display it we'll use one of the IDE's rather powerful features, the debug window. Let's get to it.

This program uses the IDE's debug window to open click Debug > Windows > Output

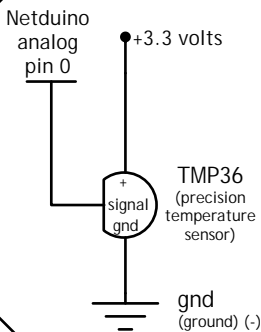
The TMP36 Datasheet:  
<http://nedx.org/TMP36>

## THE CIRCUIT:

### Parts:

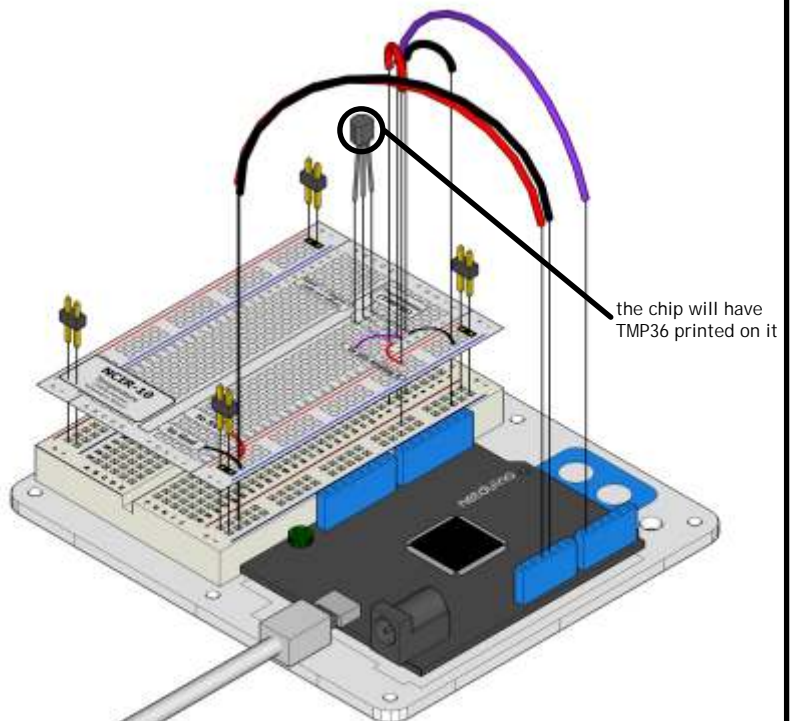


### Schematic



### The Internet

..download:..  
breadboard layout sheet  
<http://nedx.org/NBLS10>



## CODE (no need to type everything in just click)

Download from (<http://nedx.org/CODE10>)

(copy the text and paste it into an empty Netduino Project named NCIR10)

```
namespace NCIR10
{
    public class Program
    {
        static AnalogInput tempSensor = new
            AnalogInput(Pins.GPIO_PIN_A0);
        // defines the temperature sensor as
        // being connected to analog pin 0

        public static void Main()
        {
            int tempInput = 0;
            while (true) /// Do Forever ///
            {
                tempInput = tempSensor.Read();
                // Read the raw sensor value
                float volts =
                    ((float)tempInput / 1024.0f) * 3.3f;
                // The read value is from 0-
                // 1023, so this converts it
                // to a % then

                // multiplies by 3.3v to get the
                // real voltage that was read.
                float temp = (volts - 0.5f);
                // The datasheet for the sensor
                // indicates there's a 500mV (half
                // a volt) offset, this is to allow
                // it to read under 0 degrees
                temp = temp * 100;
                // Finally, every 10mV indicates 1
                // degree Celsius, so multiply by 100
                // to convert the voltage to a
                // reading
                Debug.Print(temp.ToString());
                // prints the temperature reading to
                // the debug window
                Thread.Sleep(100);
                // waits for 100 milliseconds
            }
        }
    }
}
```

## NOT WORKING? (3 things to try)

**Nothing Seems to Happen**  
This program has no outward indication it is working. To see the results you must open the Netduino IDE's debug window. (instructions on previous page)

**Temperature Value is Unchanging**  
Try pinching the sensor with your fingers to heat it up or pressing a bag of ice against it to cool it down.

**Frustration?**  
Don't worry the Netduino has a great community around it. There's always someone keen to help out on the forums. <http://nedx.org/FORU>

## MAKING IT BETTER

### Outputting voltage:

This is a simple matter of changing one line. Our sensor outputs 10mv per degree centigrade so to get voltage we simply debug the volt variable rather than temp. Replace the line:

```
Debug.Print(temp.ToString());
Debug.Print(volts.ToString());
```

### Outputting degrees Fahrenheit:

Again this is a simple change requiring only maths. To go degrees C ----> degrees F we use the formula:

```
(F = C * 1.8) + 32)
add the line
temp = (temp*1.8f) + 32f;
after
temp = temp * 100;
```

### More informative output:

Let's add a message to the debug output to make what is appearing in the Serial Monitor more informative. To do this first revert to the original code then change:

```
Debug.Print(temp.ToString());
Debug.Print(temp.ToString() + " degrees
centigrade");
```

This now outputs the string as well as the output value.

## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR10>

.:Larger Loads:.  
.:Relays:.



## WHAT WE'RE DOING:

The final circuit is a bit of a test. We combine what we learned about using transistors in NCIR03 to control a relay. A relay is an electrically controlled mechanical switch. Inside the little plastic box is an electromagnet that, when energized, causes a switch to trip (often with a very satisfying clicking sound). You can buy relays that vary in size from a quarter of the size of the one in this kit up to as big as a fridge, each capable of switching a certain amount of current. They are immensely fun because there is an element of the physical to them. While all the silicon we've played with to this point is fun sometimes you may just want to wire up a hundred switches to control something magnificent. Relays give you the ability to dream it up then control it with your Netduino. Now to using today's technology to control the past.

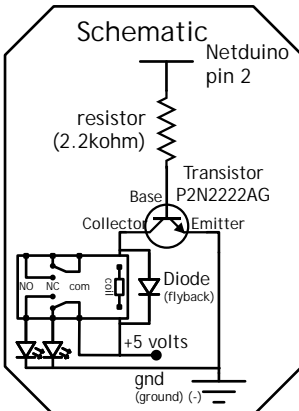
(The 1N4001 diode is acting as a flyback diode, for details on why it's there visit: <http://nedx.org/4001>)

## THE CIRCUIT:

### Parts:

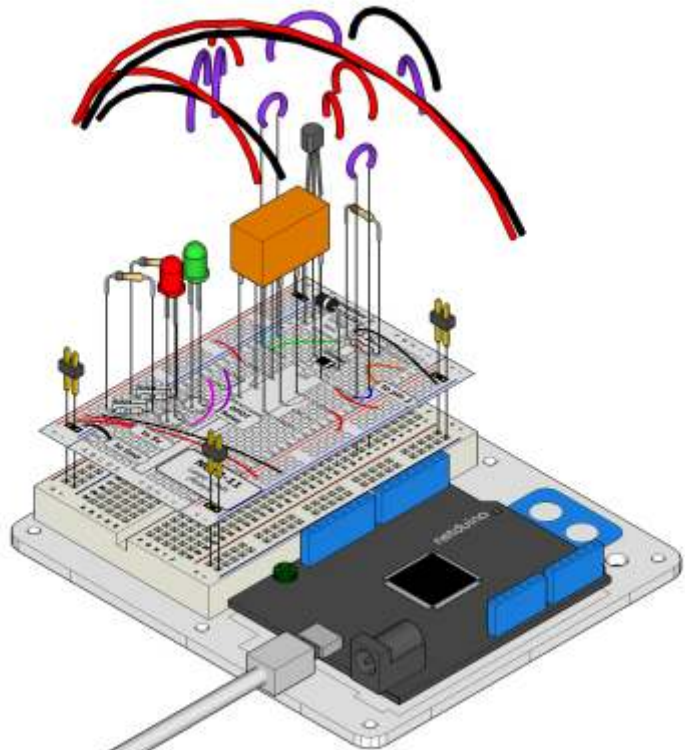
- |  |  |   |  |
|--|--|---|--|
|  NCIR-11 Breadboard Sheet x1       |  Diode (1N4001) x1                     |  Transistor P2N2222AG (TO92) x1 |  Relay (DPDT) x1 |
|  2.2k Ohm Resistor Red-Red-Red x1 |  560 Ohm Resistor Green-Blue-Brown x2 |  Green LED x1                  |  Red LED x1     |
|  2 Pin Header x4                  |  |   |  |

### Schematic



### The Internet

..download:..  
breadboard layout sheet  
<http://nedx.org/NBLS11>



## CODE (no need to type everything in just click)

Download from ( <http://nedx.org/CODE11> )  
 (copy the text and paste it into an empty Netduino Project named NCIR11)

```
namespace NCI R11
{
    public class Program
    {
        static OutputPort relayPort =
            new OutputPort(Pi ns. GPI O_PI N_D2, fal se);
        // Define the relay as being connected
        // to pin 2

        public static void Main()
        {
            while (true)          /// Do Forever ///
            {
                relayPort. Wri te(true);      // Turn the relay on
                Thread. Sl eep(1000);        // sleep for 1 second
                relayPort. Wri te(fal se);    // Turn the relay off
                Thread. Sl eep(1000);        // sleep for 1 second
            }                          /// Cl ose Forever Loop ///
        }                              /// Cl ose the Mai n() Loop ///
    }                                  /// Cl ose the Program Loop ///
}                                     /// Cl ose the Namespace Loop ///
```

## NOT WORKING? (3 things to try)

### No Clicking Sound

The transistor or coil portion of the circuit isn't quite working. Check the transistor is plugged in the right way.

### Not Quite Working

The included relays are designed to be soldered rather than used in a breadboard. As such you may need to press it in to ensure it works (and it may pop out occasionally).

### Underwhelmed?

No worries these circuits are all super stripped down to make playing with the components easy, but once you throw them together the sky is the limit.

## MAKING IT BETTER

### Watch the Back-EMF Pulse

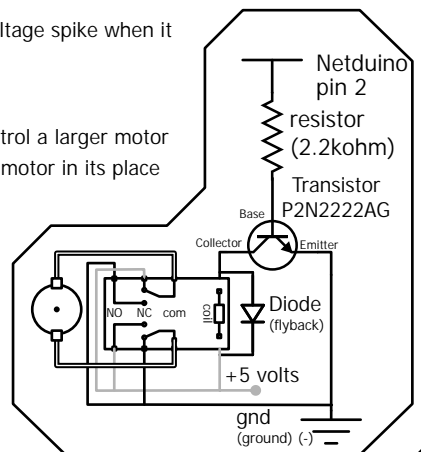
Replace the diode with an LED. You'll see it blink each time it "snubs" the coil voltage spike when it turns off.

### Controlling a Motor

In NCIR-03 we controlled a motor using a transistor. However if you want to control a larger motor a relay is a good option. To do this simply remove the red LED, and connect the motor in its place (remember to bypass the 560 Ohm resistor).

### Controlling Motor Direction

A bit of a complicated improvement to finish. To control the direction of spin of a DC motor we must be able to reverse the direction of current flow through it. To do this manually we reverse the leads. To do it electrically we require something called an h-bridge. This can be done using a DPDT relay to control the motor's direction, wire up the following circuit. It looks complicated but can be accomplished using only a few extra wires. Give it a try.



## MORE, MORE, MORE:

More details, where to buy more parts, where to ask more questions:

<http://nedx.org/NCIR11>

.:Notes:.

.:Room for a Few Notes:.

A large rectangular area with horizontal lines for writing notes. The lines are evenly spaced and cover most of the page's width and height, leaving margins at the top and bottom.

.:Notes:.

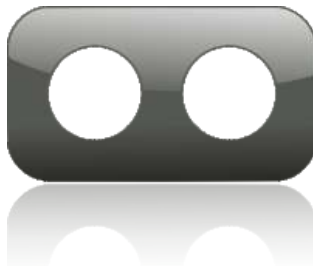
.:Room for a Few Notes:.

**NOTE**  
notes

A large rectangular area with a double-line border and horizontal ruling lines, intended for writing notes. The area is currently blank.

# netduino

[www.oomlout.com](http://www.oomlout.com)



This work is licenced under the Creative Commons Attribution-Share Alike 3.0 Unported License. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

