Getting Started with CANLight

Written by seth@mindsensors.com

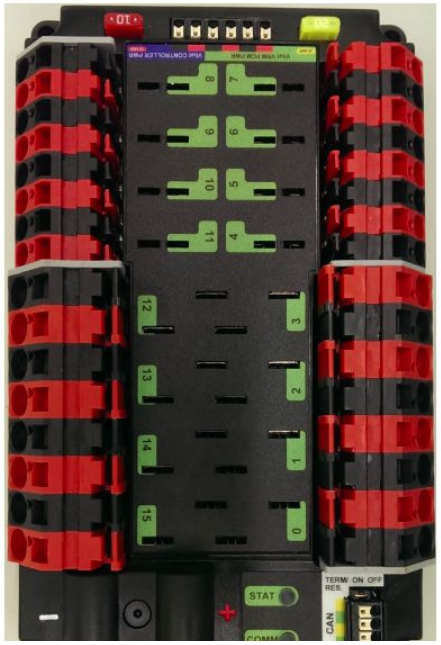
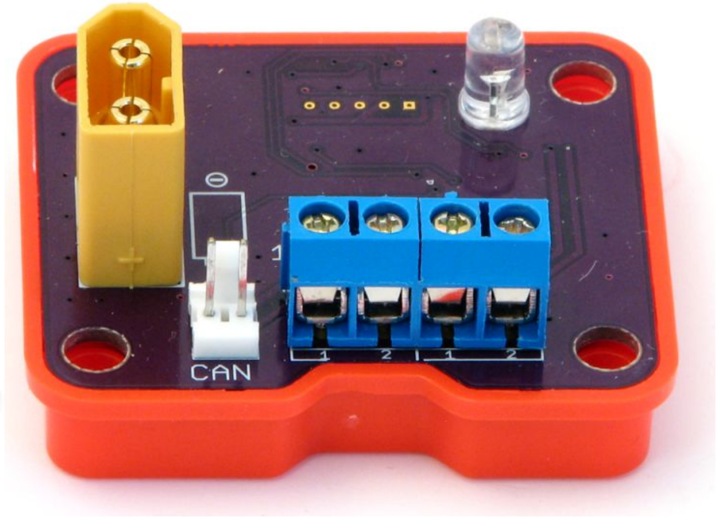
Document version: 1.3 (1/5/17)

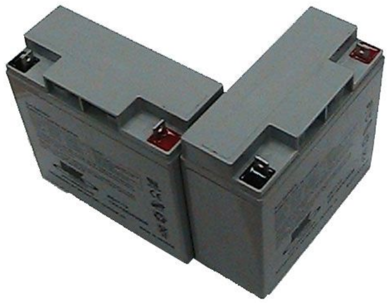
Overview

- Hardware and connections
- What is CAN?
- Setting up software
- Many example projects
- Guide to all features
- Eclipse shortcuts and useful tools
- Documentation, more resources, and getting help

Hardware Components

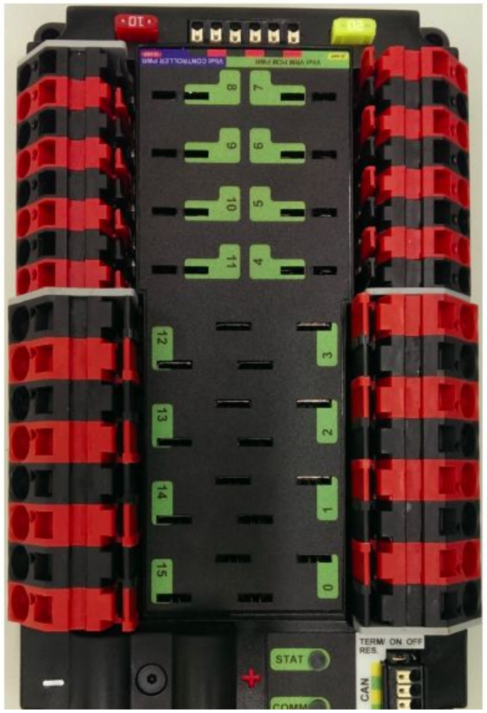
- roboRIO – the robot controller
- PDP – Power Distribution Panel
  - battery
- CANLight

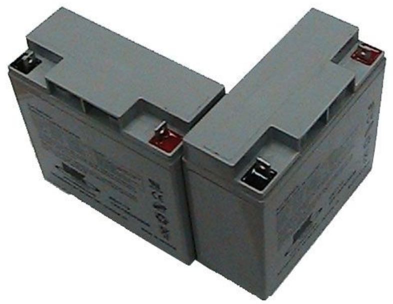





Power Distribution Panel (PDP)

- Power comes from the battery to the Power Distribution Panel
- The PDP connects power to everything else on the robot
- Two of the items it will power are the roboRIO and the CANLight.



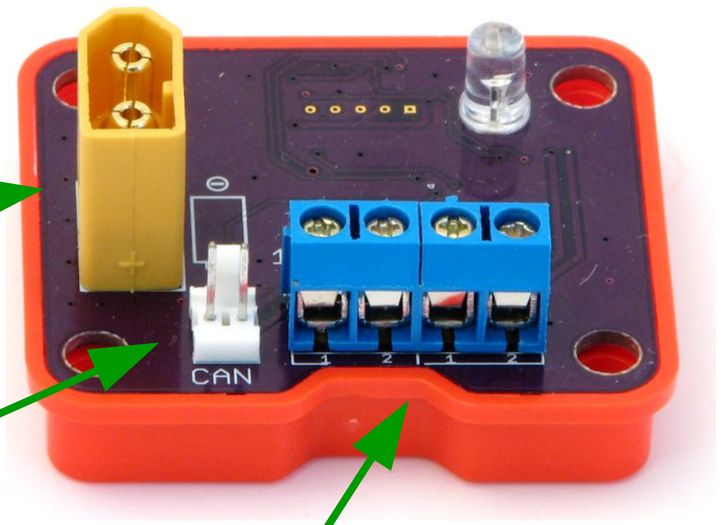


RoboRIO

- The roboRIO is your robot controller
- This is where your robot program is stored and what does all the processing
- The driver station will connect to this either through a Ethernet cable or over a wireless bridge.

CANLight

- The CANLight is a CAN-based smart controller for RGB LED strips
- It accepts power from the PDP via an XT60 connection
- It interfaces with the roboRIO over the CAN protocol
- The terminal block has four ports to connect the red, green, blue and ground wires from your RGB LED strip
  - red green blue, light-emitting diode



Connections

- Connect power from the PDP to the roboRIO and CANLight
- Connect the CANLight to the roboRIO with a CAN wire for communication
- Connect a battery to the PDP

RGB LED Strip Connection

- Of course you probably want to connect your RGB LED strip to the CANLight. Loosen the screws on the blue terminal, insert each wire, and tighten them.
- Make sure the CANLight is receiving enough amperage for the length of RGB LED strip you are using. A short strip will have no problems, with a very long one you may want to check it has enough power.
  - If the CANLight turns off when trying to set secondary colors or white, the strip is probably too long for the amount of power the CANLight is receiving.

What is CAN?

- CAN (Controller Area Network) is a communications protocol not unlike I2C or SPI you may have used in the past
- It is commonly found in vehicles
- Communication is performed over only two wires

CAN Wire

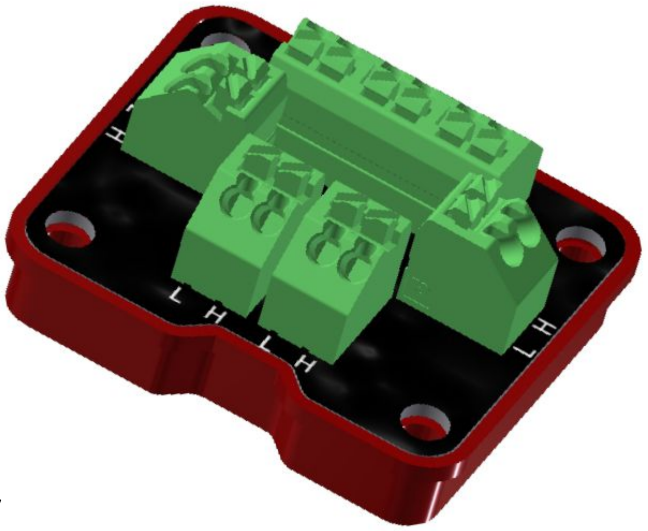
- A twisted pair of wires
- Red and black, or yellow and green
  - Red/Yellow: high
  - Black/Green: low
- Make sure polarity matches!

CAN Networks

- You can daisy-chain CAN devices
- The roboRIO has one CAN port. You can connect it to an SD540C motor controller, and connect that to a CANLight.
- However, this has negative implications for reliability. If the wire between the roboRIO and SD540C are severed (in this scenario) communication between the roboRIO and CANLight will also be terminated.
- While that might not be a huge issue in this case (if the lights go off the robot can still run), if multiple SD540C motor controllers are attached this problem may be more significant.

Solution to the Reliability Problem

- There are other network typologies
- An appealing option is to use a CAN splitter
- With this you connect the roboRIO and all CAN devices to the splitter



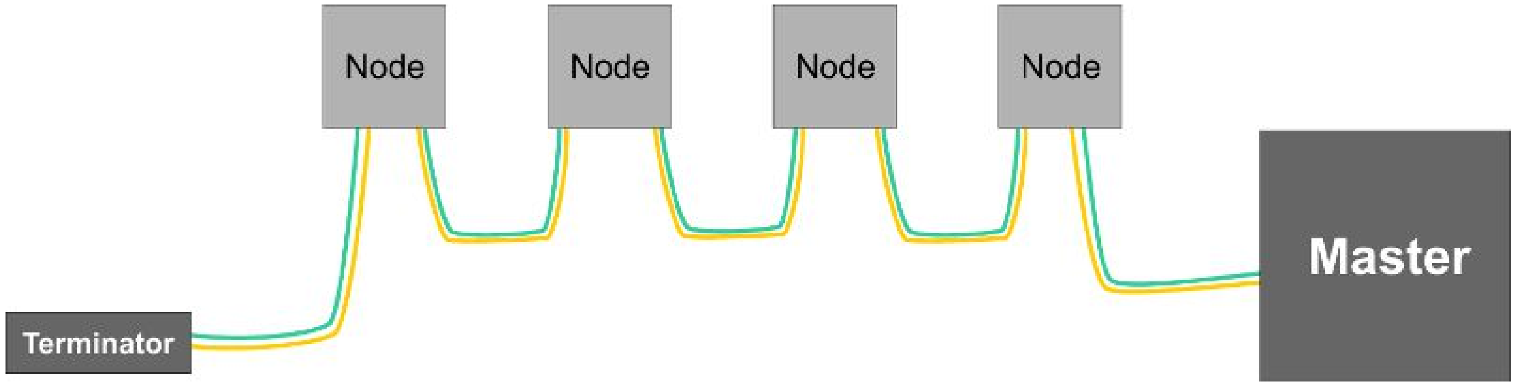
- Product name: CANSplitter
  - http://www.mindsensors.com/frc/184-splitter-for-can-network

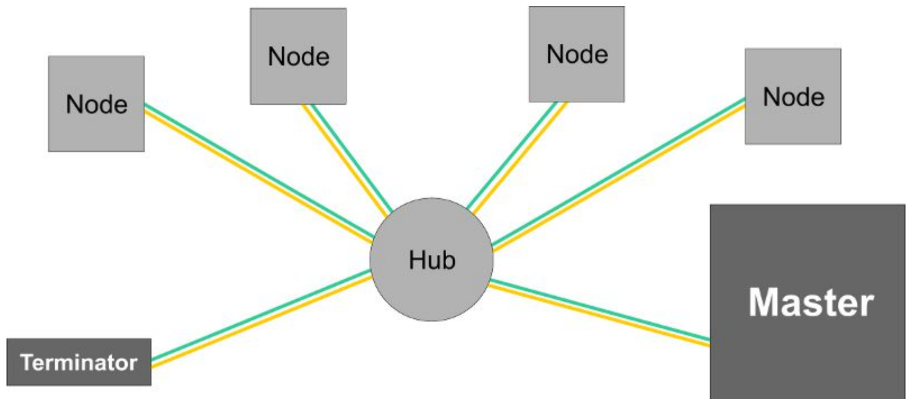
The 120Ω Resistor

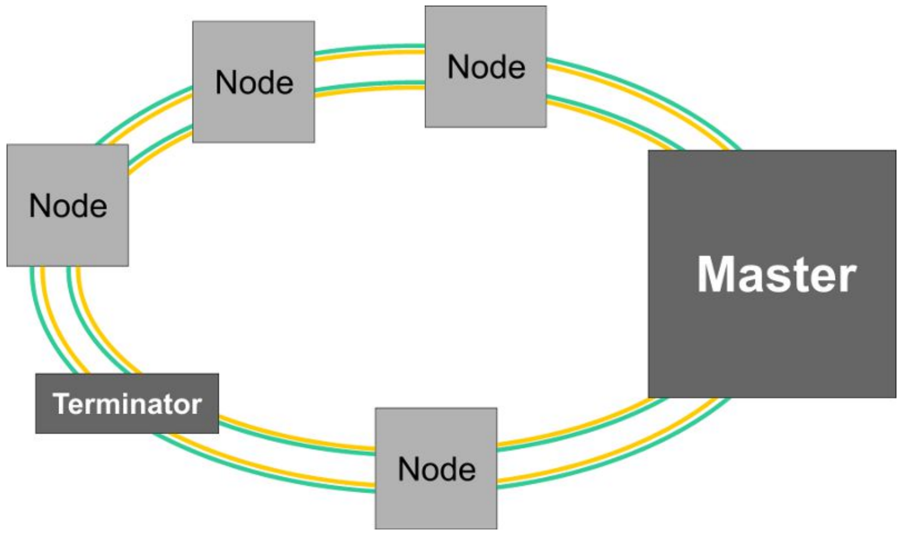
- The CAN protocol requires a 120 ohm resistor on either end of the network in a chain. The roboRIO has one built in, so if you are daisy-chaining devices then you should end the chain with a second 120 ohm resistor.
- If you are using mindsensors.com's CAN Splitter, it already has a resistor built in (like the roboRIO)
  - You can activate or bypass it with a jumper.

Even Mode Information on CAN

- http://www.mindsensors.com/content/86-can-and-its-topology







Finishing Connections

- With those few connections complete you should be able to move on to programming!
- The last physical connection is connecting the roboRIO to your programming computer
- The roboRIO has both a USB and an Ethernet port for networking

Wireless

- Wireless bridge
  - the roboRIO connects to a small router on the robot
  - another router connects to the driver station computer
  - the routers are configured to act as a direct connection
- You may consider using a hard-wired connection for reliability when programming

RoboRIO Routing

- You will need a host name to connect to your roboRIO
- Host names (replace #### with your team number)
  - roboRIO-####-FRC.local
  - 10.##.##.20...
  - USB: 172.22.11.2
- More information
  http://wpilib.screenstepslive.com/s/4485/m/13503/l/242608-roborio-networking

Programming

- Let's start programming! Here's an outline of the first-time setup steps
  - Install the FRC 2017 Update Suite
  - Image your roboRIO
  - Download and install Eclipse Mars (4.5)
  - Add WPILib Robot Development plugins to Eclipse
  - Add the mindsensors.com FRC Library
  - Create your first project
- WPI provides detailed tutorials on all of this http://wpilib.screenstepslive.com/s/4485/m/13503

FRC 2017 Update Suite

- This contains a lot of useful (and required) software this includes
  - LabVIEW and FRC Robot Simulator
  - FRC Driver Station
    - This is what you will use to control and enable your robot after writing a program for it
  - roboRIO Imaging Tool
    - Use this to put the 2017 FRC image on your roboRIO
- Check your Kit of Parts for a DVD and National Instruments' website

Eclipse

- Install Eclipse Mars
- Installing the WPILib plugins
  - Help → Install New Software...
  - Work with: http://first.wpi.edu/FRC/roborio/release/eclipse/
  - Check WPILib Robot Development, containing Robot C++ Development and Robot Java Development
  - Click next, finish installation process, restart Eclipse

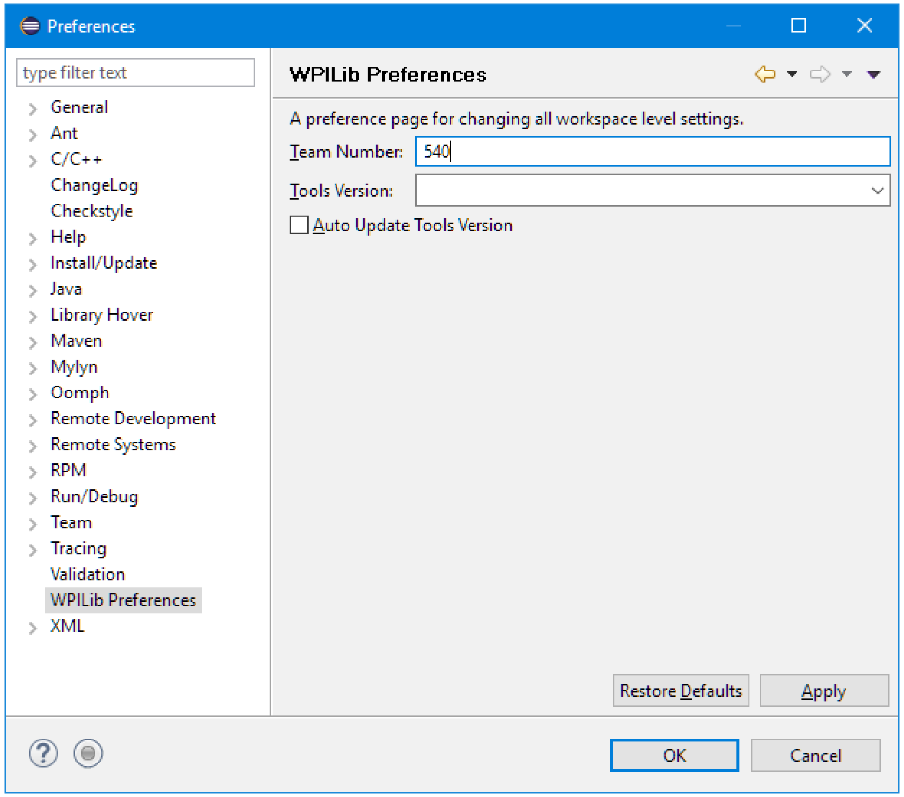
mindsensors.com FRC Library

- Download the zip at http://mindsensors.com/largefiles/FIRST/mindsensors.zip
- Copy the user folder to C:\Users\username\wpilib\user
  - If it prompts for existing files, replace them with new ones
- Find detailed instructions on our blog post http://www.mindsensors.com/blog/how-to/how-to-use-sd540c-and-canlight-with-roborio
- Documentation available at

  http://www.mindsensors.com/reference/FRC/html/Java/
  http://www.mindsensors.com/reference/FRC/html/C++/

Eclipse Team Number Configuration

- If you haven't already set your team number, do so now before you forget
  - Window → Preferences
  - WPILib Preferences
  - Team Number: ####



Preferences

type filter text

WPILib Preferences

A preference page for changing all workspace level settings.

Team Number: 540

Tools Version:

☐ Auto Update Tools Version

Install Java

- If you are using Java, install Java on the roboRIO
- http://wpilib.screenstepslive.com/s/4485/m/13503/l/599747-installing-java-8-on-the-roborio-using-the-frc-roborio-java-installer-java-only
- https://goo.gl/xL7fZu

First Project

- Let's create a project to start experimenting with!
- New project
  - *Ctrl+N* or *File → New → Other...*
  - Open *WPILib Robot C++/Java Development* folder
  - Select *Example Robot C++/Java Project*
  - *Getting Started* template
  - Click Next to rename project or Finish

Open Robot.cpp / Robot.java

- In the Project Explorer window, open the project you just created (named Getting Started by default)
  - Don't see the Project Explorer?
    Window → Show View → Project Explorer
- Open the *src* folder
  - C++
    - Double-click *Robot.cpp*
  - Java
    - Open *org.usfirst.frc.team####.robot* package
    - Double-click *Robot.java*
- This is the file we will be replacing with some examples

Hello World □ □

C++

```cpp
#include "WPILib.h"

class Robot: public IterativeRobot {

private:
    void RobotInit() {
        printf("Hello, world!\n");
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```java
package org.usfirst.frc.team####.robot;

import edu.wpi.first.wpilibj.IterativeRobot;

public class Robot extends IterativeRobot {

    public void robotInit() {
        System.out.println("Hello, world!");
    }

}
```

Hello World

C++

```
#include "WPILib.h"

class Robot: public IterativeRobot {

private:
    void RobotInit() {
        printf("Hello, world!\n");
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```
package org.usfirst.frc.team####.robot;

import edu.wpi.first.wpilibj.IterativeRobot;

public class Robot extends IterativeRobot {

    public void robotInit() {
        System.out.println("Hello, world!");
    }

}
```

The example project comes with a lot of example code in your robot (robot.java/Robot.cpp) file. Replace it with the code above. First we're just going to print "Hello, world!" to the console, classic.

In C++ we include the WPILib.h header file. In Eclipse you can select it and press F3 (or right-click → Open Declaration) to see what this file is. It just includes the rest of the WPI robotic library for FRC. Here we just need the IterativeRobot template, but we will be making use of the other components in the future.

In our Robot.cpp file we declare a Robot class that inherits from IterativeRobot. Ask one of your friendly upperclassmen programmers if you haven't used inheritance in object-oriented programming before. Everything that an IterativeRobot will have, our Robot will now have. Again, try F3 to learn what an IterativeRobot is (or wait for the next slide).

In our Robot class we define a private method RobotInit which will print "Hello, world!"

We end our file with a macro which will do some magic to make things simple for us (actually it's not magic, remember F3, you don't need to worry about this part though).

Java! Hello you highly-caffeinated friends. The description of C++ over there can be helpful, but we'll go over the Java code specifically. The first line is a package declaration. This won't be of much significance now (besides replacing #### with your team number!) but basically you have your own little package just for you to put your robot code.

Next we import the IterativeRobot class. Then we create a Robot class that extends IterativeRobot. Click IterativeRobot and press F3 (or right-click → Open Declaration) to see it's source code. Ask a helpful teammate if you haven't used object-oriented programming before.

Inside our class we override the robotInit method. Why override? Well, take a look at IterativeRobot. It has a method with exactly the same method signature (access specifier: *public*, return type: *void*, name: *robotInit*, parameters: *[none]*). All this method does though is print a message. While we're not doing much more right now, we still want to run our own code instead, so we're overriding it.

Deploying

- Right click your *project* folder (Getting Started or similar, a blue folder) in the Project Explorer (left side panel). If you don't see it then go to Window → Show View → Project Explorer.
- Mouse-over the Run As submenu about two-thirds down. Eclipse may freeze as moment here. Click "WPILib C++/Java Deploy" and again, Eclipse may freeze for several seconds.
- If your computer can communicate with the roboRIO and you've set your team number, it should connect and deploy your new code!

Driver Station

- You should be able to find the Driver Station in C:\Program Files (x86)\FRC Driver Station\DriverStation.exe, but you took care of that when you followed the *Installing the FRC 2016 Update Suite (All Languages)*, right?
- On the left panel there is a small gear in the upper-right corner. Click this and select View Console. You should see "Hello, world!" in green from your robot!

Introduction to IterativeRobot

- The IterativeRobot class is a nice template to help you get started quickly. It has eight important methods you can use. They start with *robot*, *autonomous*, *teleop*, or *disabled* and end with *Init* or *Periodic*.
  - robotInit()
  - robotPeriodic()
  - autonomousInit()
  - autonomousPeriodic()
  - teleopInit()
  - teleopPeriodic()
  - disabledInit()
  - disabledPeriodic()
- Note: method names begin with a capital letter in C++ but lower-case in Java

Introduction to IterativeRobot

Autonomous methods will be called, as you might have guessed, in autonomous. *AutonomousInit* is run once when autonomous begins and *AutonomousPeriodic* is called repeatedly and rapidly as long as autonomous is enabled.

- The same is true for teleop. If you use would like to use test mode then those methods exist as well.
- *RobotInit* will be run once when your robot start up, and *RobotPeriodic* will always run repeatedly. *DisabledInit* runs once when your robot becomes disabled (for any reason), and *DisabledPeriodic* runs as long as your robot is disabled.

Init vs. Periodic Methods

- To be clear, the __*Periodic* methods are called repeatedly and (hopefully) many times a second!
- One-time setup/initialization code goes in __*Init*, repeated code goes in __*Periodic*

Lighting the CANLights

C++

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
    }

    void AutonomousInit() {
        lights->ShowRGB(255, 255, 255);
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```java
package org.usfirst.frc.team####.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
    }

    public void autonomousInit() {
        lights.showRGB(255, 255, 255);
    }

}
```

Lighting the CANLights

- Now for real let's light up the CANLight! We will build on what we learned earlier in Hello World. First we are including/importing an additional file.
- In C++ we *#include "mindsensors.h"* but you can also include only *CANLight.h* in this example. CANLight is in the mindsensors namespace, so we use a *using* statement so we don't have to put *mindsensors::* each time we type CANLight. Ask a teammate about the namespace resolution operator!
- In Java you can *include com.mindsensors.*;* or just *com.mindsensors.CANLight* for this example.

Lighting the CANLights

- The next new addition is a *lights* attribute to our *Robot* class. It's type is a *CANLight* object, but *lights* doesn't have a value yet.
- Now in *RobotInit* we construct a new CANLight object for *lights*. Ask a friendly teammate programmer about object-oriented programming if this is new to you.
- We add an *AutonomousInit* method and use it to call the *ShowRGB* method of our *lights* object.
- Bonus: what will this program do?

Lighting the CANLights

- This program will make the CANLight illuminate a white color on the RGB LED strip when autonomous starts. Good job!
  - Try setting a different color than white. Maybe even set another color when teleoperated mode starts!
- You may notice the lights will stay on even when autonomous ends. The CANLight will continue with the last command you gave it until you tell it otherwise. How might you turn the lights off when autonomous ends?
  Answer on the next slide.

Putting Out the CANLights

C++

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
    }

    void AutonomousInit() {
        lights->ShowRGB(255, 255, 255);
    }

    void DisabledInit() {
        lights->ShowRGB(0, 0, 0);
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```java
package org.usfirst.frc.team####.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
    }

    public void autonomousInit() {
        lights.showRGB(255, 255, 255);
    }

    public void disabledInit() {
        lights.showRGB(0, 0, 0);
    }

}
```

Built-In Memory

- The CANLight has a set of 8 registers
- register: one of a small set of data
- Each register has a red, green, and blue value, along with a duration
- The color components may each have a value between 0 and 255
- Duration is a number of seconds between 0.0 and 2.55 seconds
  - ex: 0.5 is half a second

Registers

- This is a visual representation of the register system

	0	1	2	3	4	5	6	7
time	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
red	0	255	0	0	255	0	255	255
green	0	0	255	0	255	255	0	255
blue	0	0	0	255	0	255	255	255

Using the Registers

- So what are the registers for?
- They allow you to create more involved light sequences without any advanced programming!
- Each register will have a default value when the CANLight turns on (receives power)
- Want to find out what these default values are?

ShowRegister

- ShowRegister accepts an index, 0 to 7, of which register to show

C++

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        lights->ShowRegister(1);
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```java
package org.usfirst.frc.team####.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        lights.showRegister(1);
    }

}
```

WriteRegister

- Want to change what's in a register? WriteRegister takes the index to write to, the duration, and red, green, and blue intensities

C++

```
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        lights->WriteRegister(1, 0.5, 255, 0, 127);
    }

};

START_ROBOT_CLASS(Robot)
```

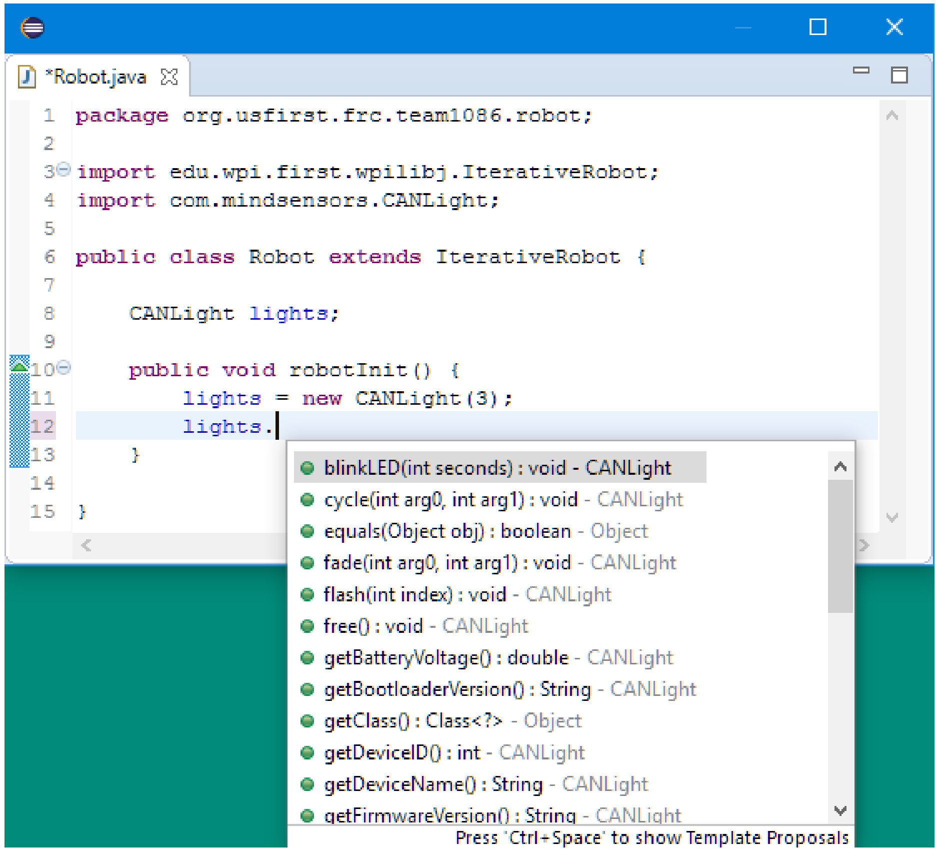
Java

```
package org.usfirst.frc.team####.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        lights.writeRegister(1, 0.5, 255, 0, 127);
    }

}
```

Aside: Tooltips and Autocomplete

- What if you forgot what parameters *WriteRegister* accepts, or what order to put them?
- You could check the documentation, we'll go over this later
- What if you don't want to leave Eclipse?

Ctrl+space

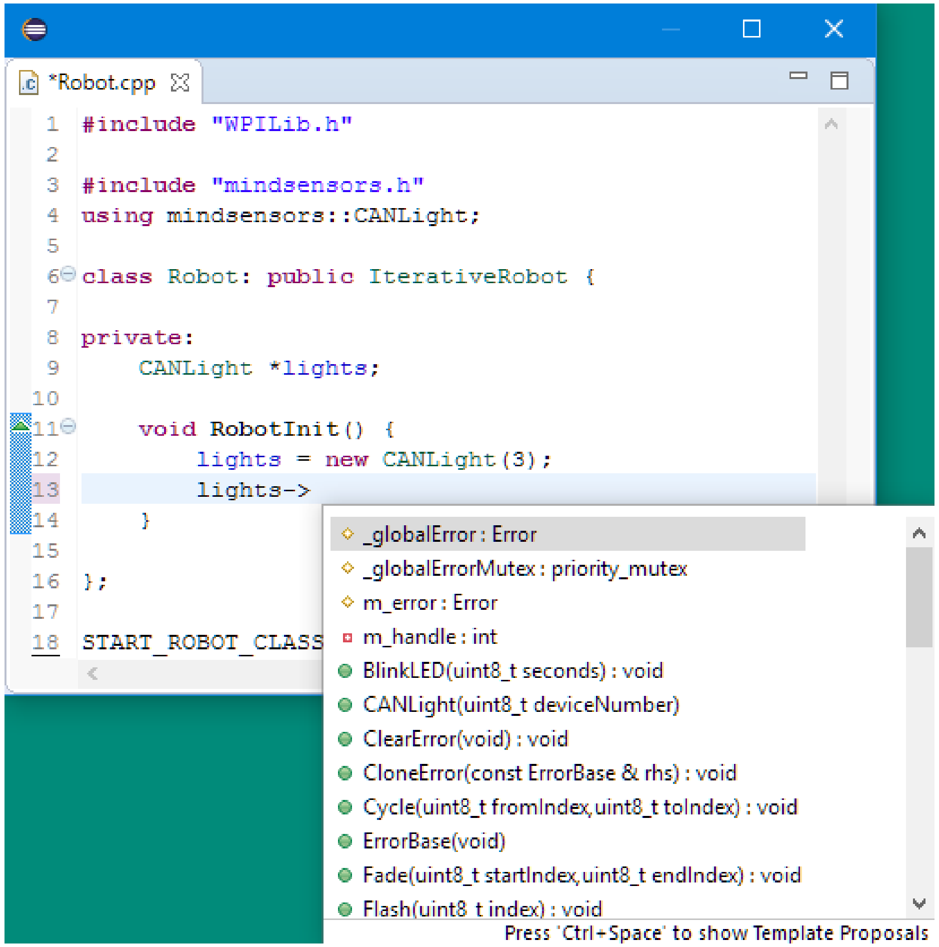


*Robot.java

```java
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        lights.
    }

}
```

```
blinkLED(int seconds) : void - CANLight
cycle(int arg0, int arg1) : void - CANLight
equals(Object obj) : boolean - Object
fade(int arg0, int arg1) : void - CANLight
flash(int index) : void - CANLight
free() : void - CANLight
getBatteryVoltage() : double - CANLight
getBootloaderVersion() : String - CANLight
getClass() : Class<?> - Object
getDeviceID() : int - CANLight
getDeviceName() : String - CANLight
getFirmwareVersion() : String - CANLight
```

Press 'Ctrl+Space' to show Template Proposals



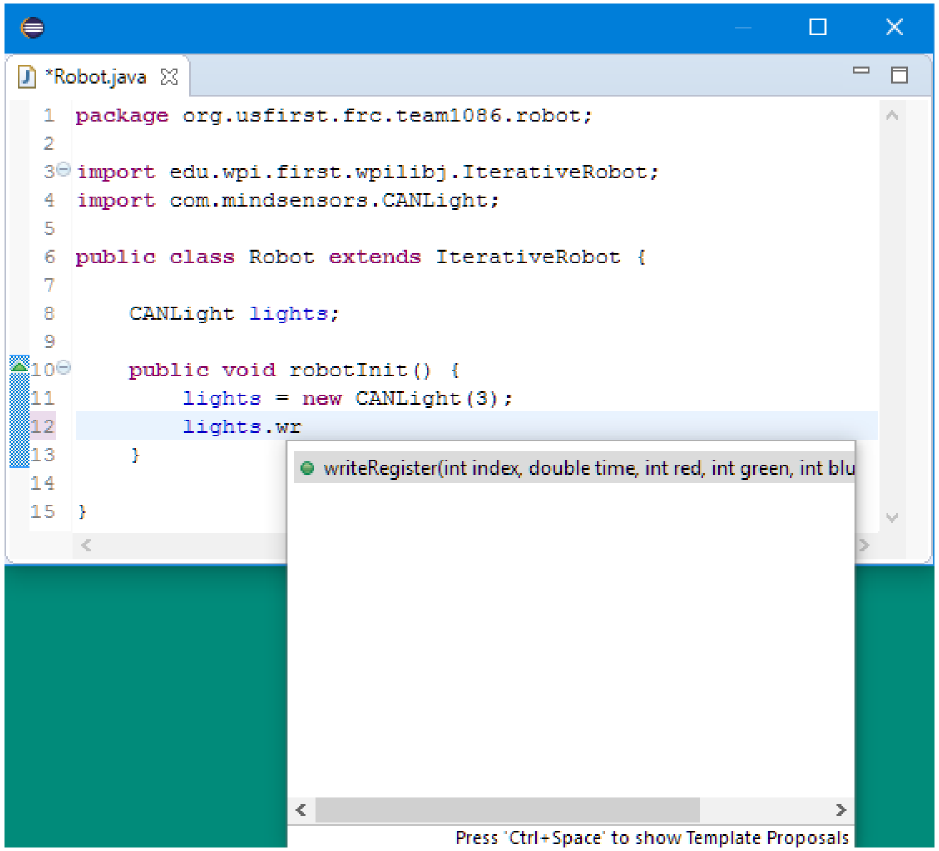
*Robot.cpp

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        lights->
    }

};

START_ROBOT_CLASS
```

```
_globalError : Error
_globalErrorMutex : priority_mutex
m_error : Error
m_handle : int
BlinkLED(uint8_t seconds) : void
CANLight(uint8_t deviceNumber)
ClearError(void) : void
CloneError(const ErrorBase & rhs) : void
Cycle(uint8_t fromIndex, uint8_t toIndex) : void
ErrorBase(void)
Fade(uint8_t startIndex, uint8_t endIndex) : void
Flash(uint8_t index) : void
```

Press 'Ctrl+Space' to show Template Proposals

Start typing to narrow search

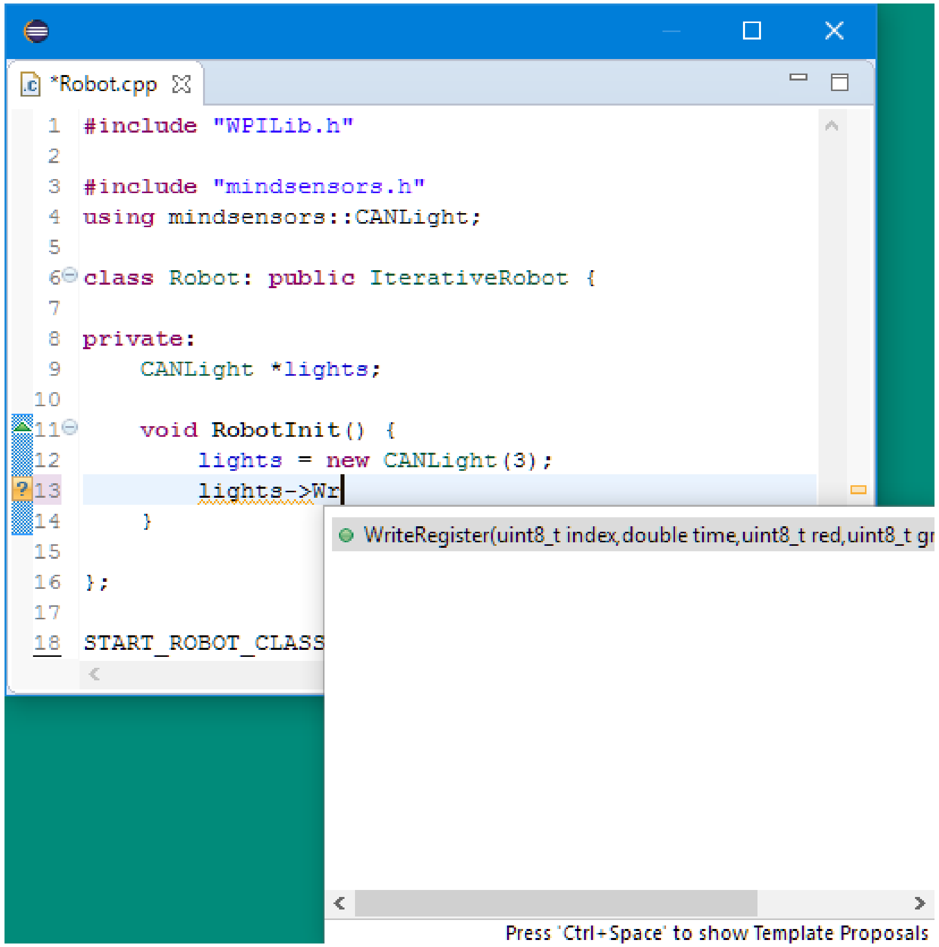


*Robot.java

```
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        lights.wr
    }

}
```

● writeRegister(int index, double time, int red, int green, int blu

Press 'Ctrl+Space' to show Template Proposals



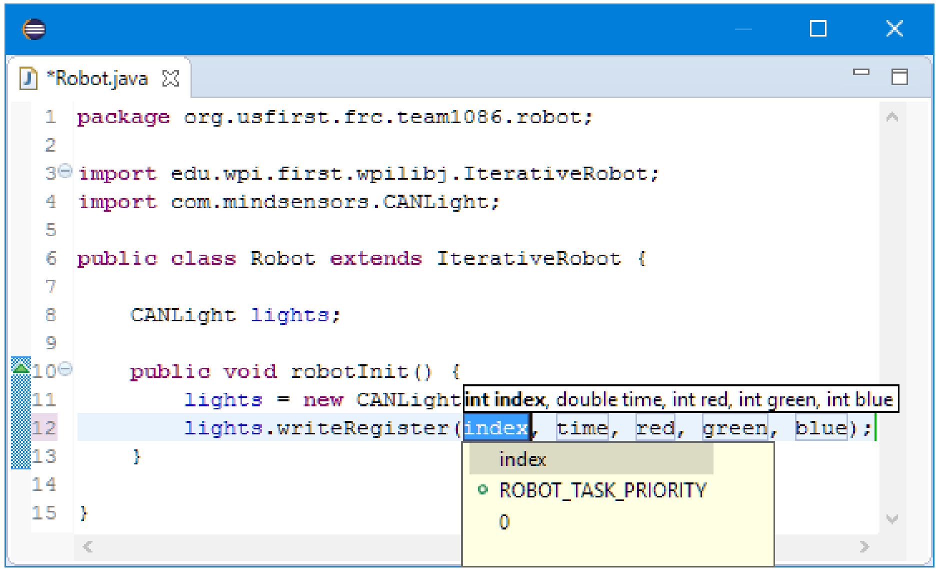
*Robot.cpp

```
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        lights->Wr
    }

};

START_ROBOT_CLASS
```

● WriteRegister(uint8_t index,double time,uint8_t red,uint8_t gr

Press 'Ctrl+Space' to show Template Proposals

Enter

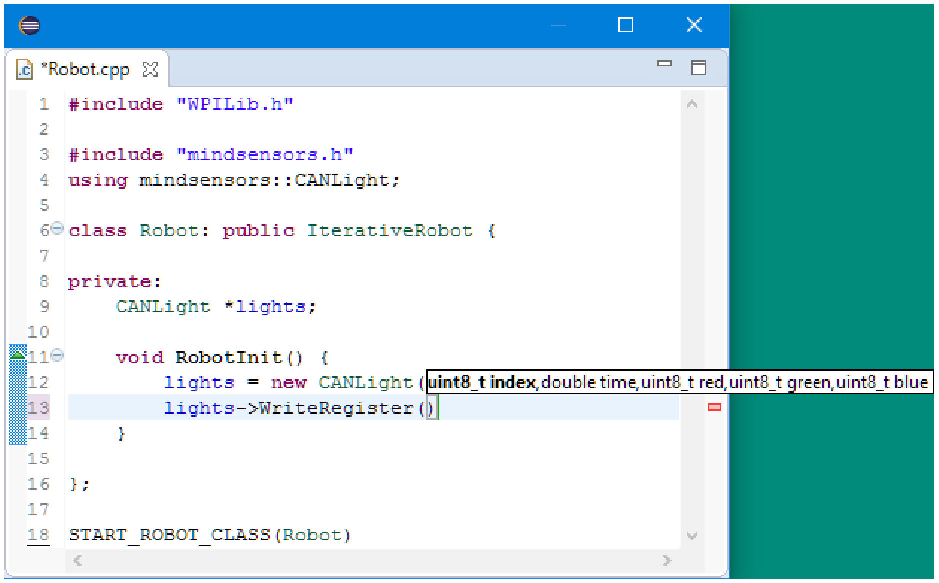


*Robot.java

```java
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight
        lights.writeRegister(index, time, red, green, blue);
    }

}
```

int index, double time, int red, int green, int blue

index
ROBOT_TASK_PRIORITY
0

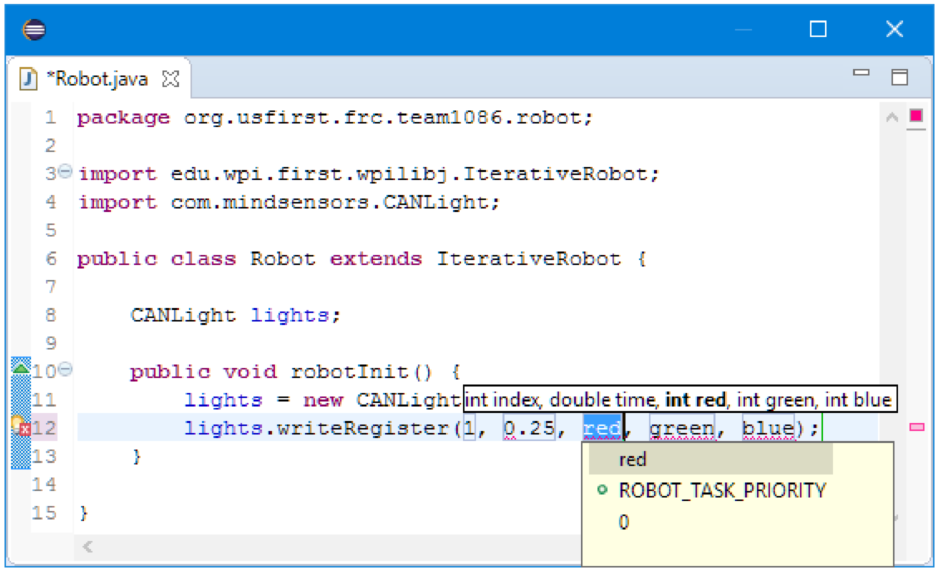


*Robot.cpp

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight (
        lights->WriteRegister()
    }

};

START_ROBOT_CLASS(Robot)
```

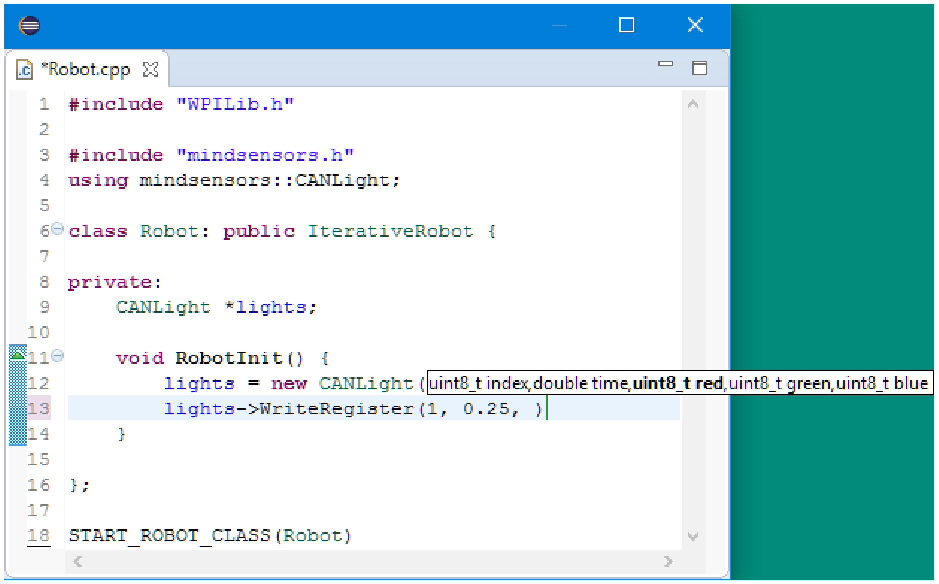
uint8_t index,double time,uint8_t red,uint8_t green,uint8_t blue

Type a value, tab (Java)



*Robot.java

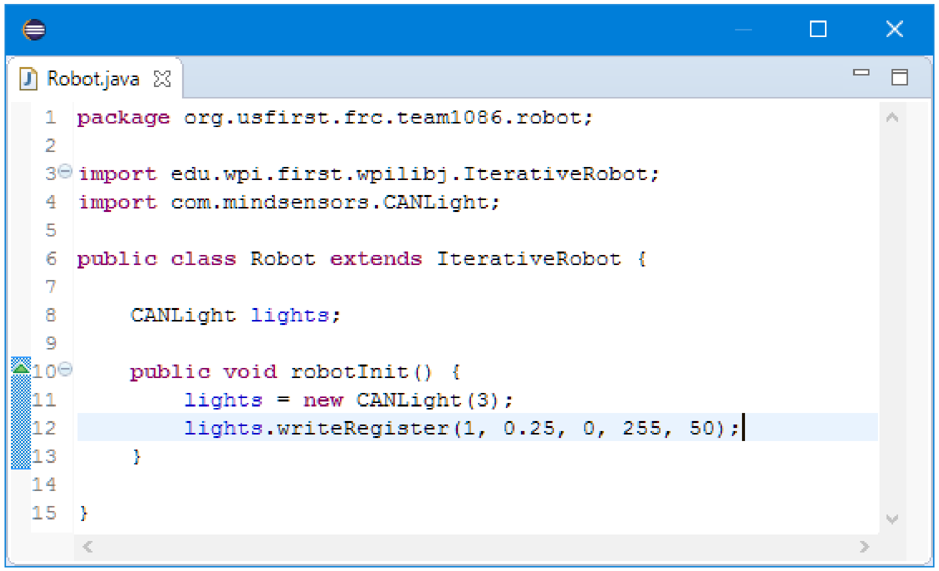
```
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(int index, double time, int red, int green, int blue
        lights.writeRegister(1, 0.25, red, green, blue);
    }

}
```



*Robot.cpp

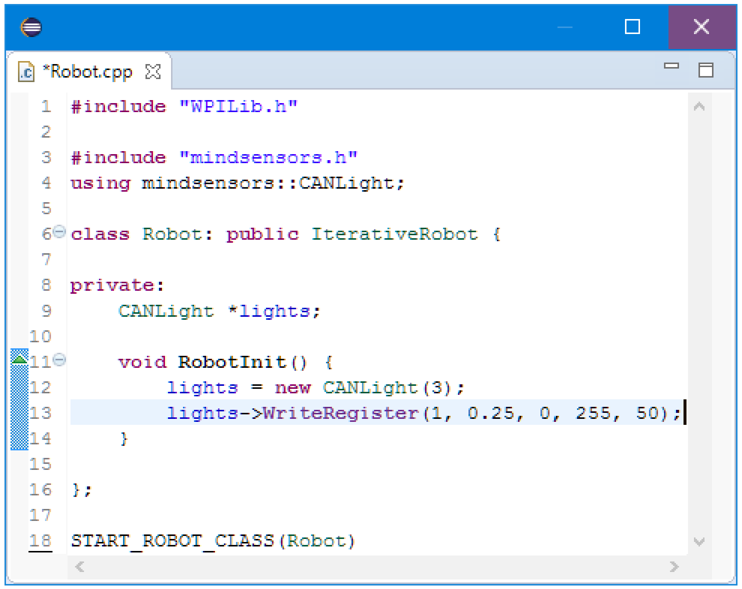
```
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(uint8_t index,double time,uint8_t red,uint8_t green,uint8_t blue
        lights->WriteRegister(1, 0.25, )
    }

};

START_ROBOT_CLASS(Robot)
```

Done! (enter to complete)



Robot.java

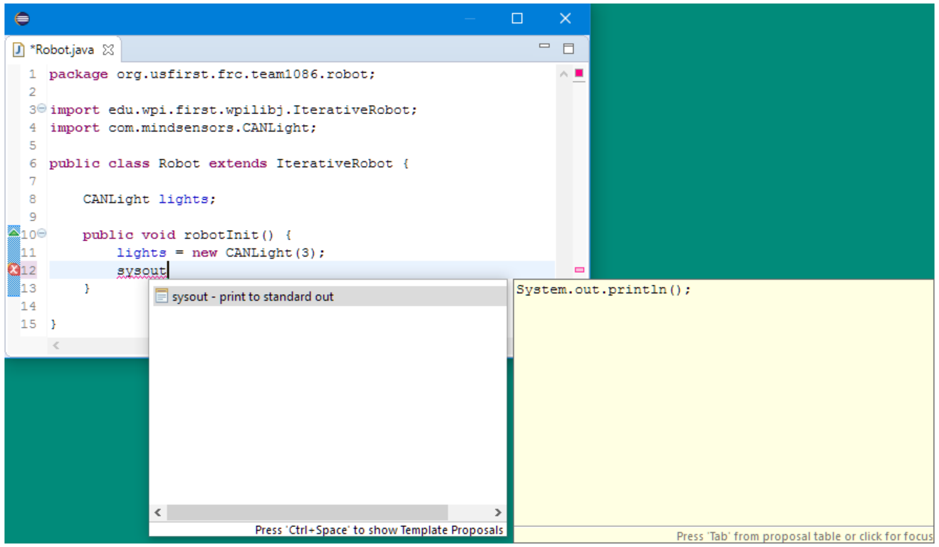
```java
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        lights.writeRegister(1, 0.25, 0, 255, 50);
    }

}
```



*Robot.cpp

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        lights->WriteRegister(1, 0.25, 0, 255, 50);
    }

};

START_ROBOT_CLASS(Robot)
```

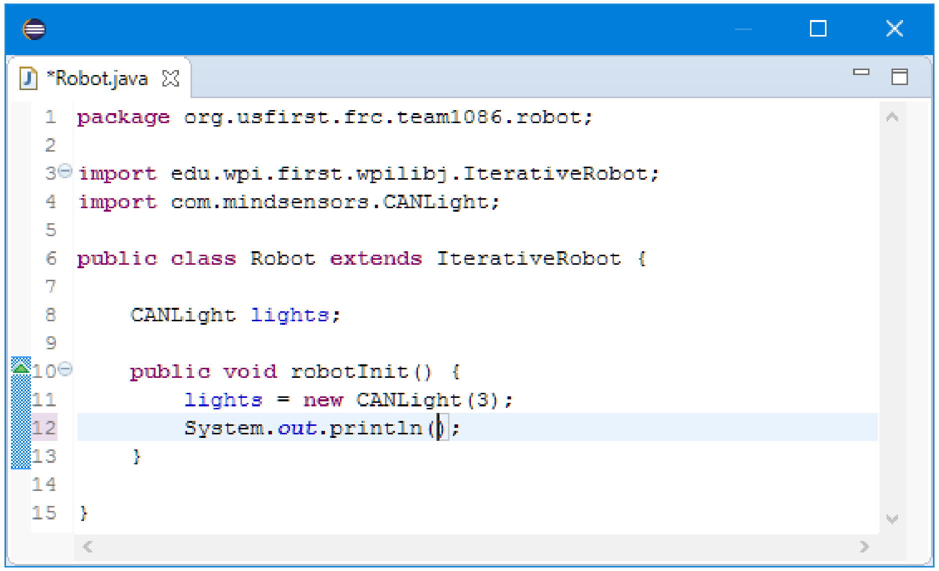
More: sysout



*Robot.java
sysout - print to standard out
Press 'Ctrl+Space' to show Template Proposals
Press 'Tab' from proposal table or click for focus

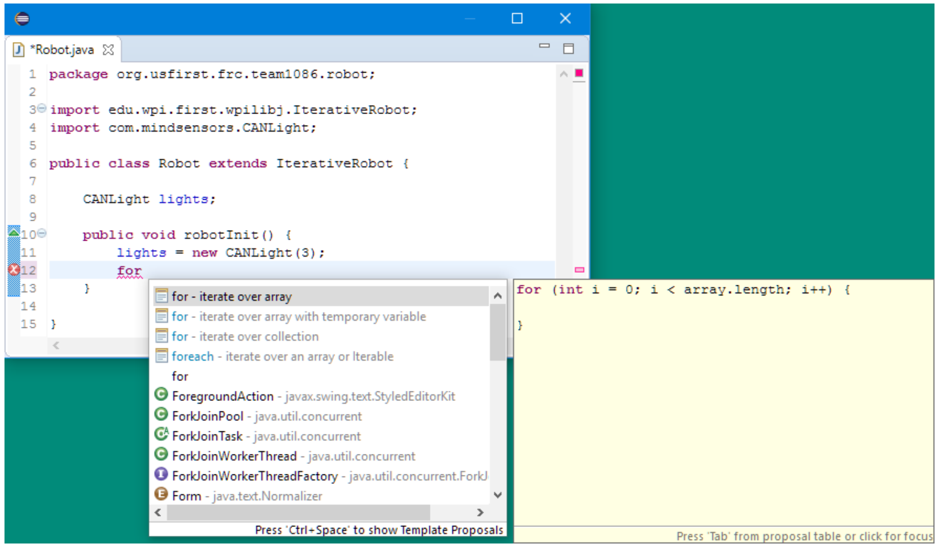
```
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        sysout
    }

}
```

```
System.out.println();
```



```
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        System.out.println();
    }

}
```

More: for



*Robot.java

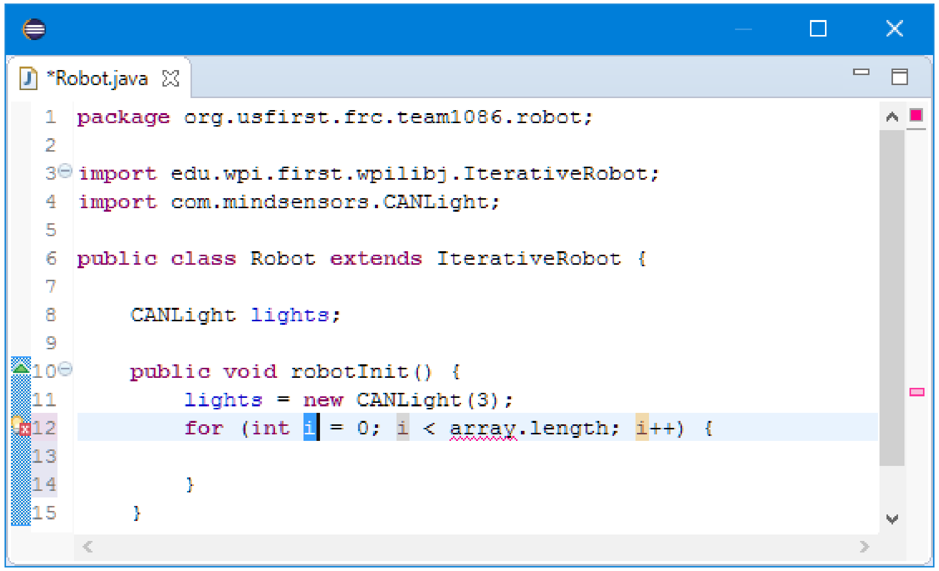
```
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        for
    }

}
```

```
for - iterate over array
for - iterate over array with temporary variable
for - iterate over collection
foreach - iterate over an array or Iterable
for
ForegroundAction - javax.swing.text.StyledEditorKit
ForkJoinPool - java.util.concurrent
ForkJoinTask - java.util.concurrent
ForkJoinWorkerThread - java.util.concurrent
ForkJoinWorkerThreadFactory - java.util.concurrent.ForkJ
Form - java.text.Normalizer
```

Press 'Ctrl+Space' to show Template Proposals

```
for (int i = 0; i < array.length; i++) {

}
```

Press 'Tab' from proposal table or click for focus



*Robot.java

```
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        for (int i = 0; i < array.length; i++) {

        }
    }
```

Recap

- So far you know how to
  - set a red, green, blue value
  - store colors to registers
  - load colors from registers
- But wait, what about the cool features using registers?

Flash

- Turns the CANLight on with the color in a register
- leaves it on for the *duration* in that register
- turns it off for that duration
- Repeat!

C++

```
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        lights->Flash(1);
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```
package org.usfirst.frc.team####.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        lights.flash(1);
    }

}
```

Flash

- You don't need to write any timing code in your robot program
- Just call *flash* and it will keep flashing until you give it another command!
- Try out some different colors and durations

Registers Default Values

- Let's say you want to find out what the default register values are
- You could also check the documentation or the second slide on registers but...

C++

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        for (uint8_t i = 0; i < 8; i++) {
            lights->ShowRegister(i);
            Wait(1.0);
        }
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```java
package org.usfirst.frc.team1086.robot;

import edu.wpi.first.wpilibj.IterativeRobot;
import edu.wpi.first.wpilibj.Timer;

import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

    CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        for (int i = 0; i < 8; i++) {
            lights.showRegister(i);
            Timer.delay(1);
        }
    }

}
```

Wait...

- This program will cycle through each of the 8 registers (indexed 0 to 7) and wait a second after each
- Doesn't this seem like too much code for such a simple task?

Cycle

C++

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        lights->Cycle(0, 7);
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```java
package org.usfirst.frc.team####.robot;

import
edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        lights.cycle(0, 7);
    }

}
```

- How about that!

Cycle

- Again, no need for any timing code of your own!
- Cycle takes 2 parameters
  - index to cycle from
  - index to cycle to
- It will continue to cycle through these until it is given another command
- Each color will be shown for the duration in that register
- Ex: `lights.cycle(1, 3);`
  - 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2...

Fade

C++

```cpp
#include "WPILib.h"

#include "mindsensors.h"
using mindsensors::CANLight;

class Robot: public IterativeRobot {

private:
    CANLight *lights;

    void RobotInit() {
        lights = new CANLight(3);
        lights->Fade(1, 3);
    }

};

START_ROBOT_CLASS(Robot)
```

Java

```java
package org.usfirst.frc.team####.robot;

import
edu.wpi.first.wpilibj.IterativeRobot;
import com.mindsensors.CANLight;

public class Robot extends IterativeRobot {

CANLight lights;

    public void robotInit() {
        lights = new CANLight(3);
        lights.fade(1, 3);
    }

}
```

Fade

- Smooth transition between colors
- The duration in each register is how long it will take to fade *from* that color to the next

Memory Lifespan and Reset

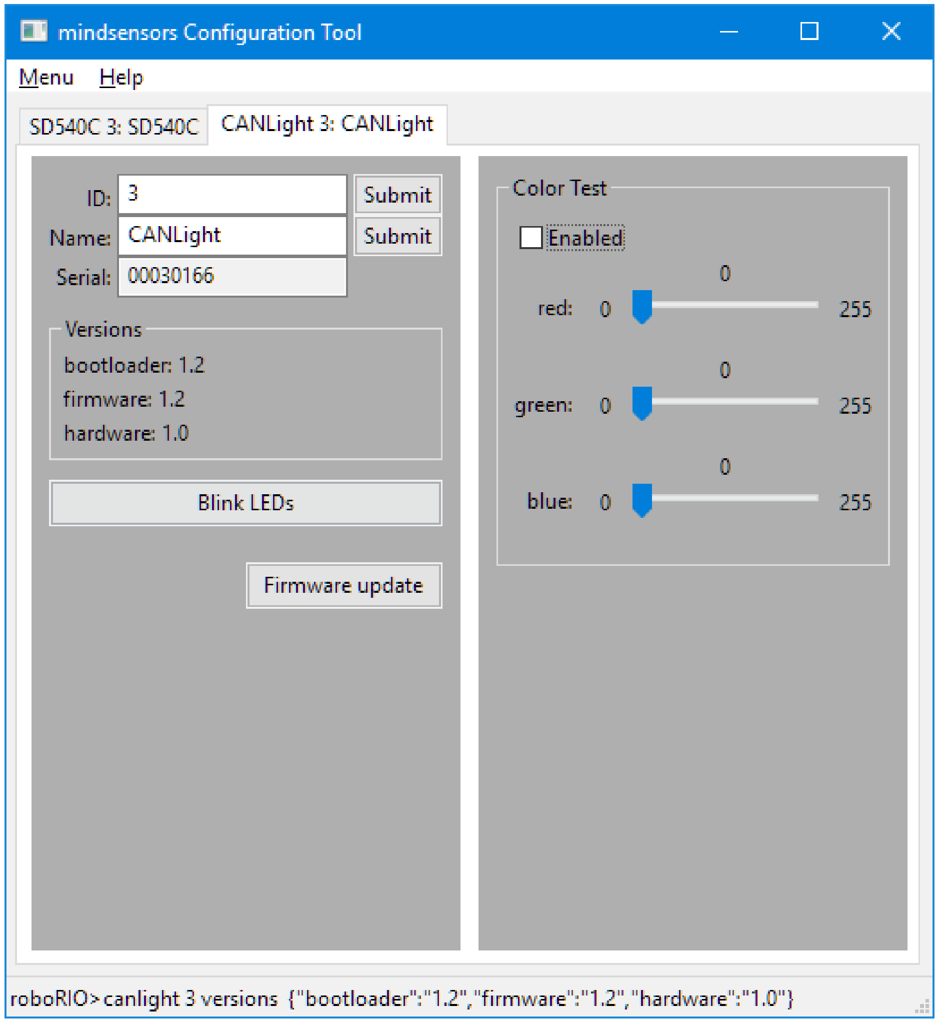
- When a CANLight loses power, its registers return to their default values
- Your robot might briefly lose power in a match
- A good place to call *WriteRegister* is in *RobotInit*, or *AutonomousInit* and *TeleopInit* if you have different color patterns for each mode
- You can call (Java) *.reset()* or (C++) *->Reset()* to return all registers to their default values

mindsensors Configuration Tool

- You can use this tool to change the ID and test colors

http://www.mindsensors.com/blog/how-to/using-the-mindsensors-configuration-tool

Color Test



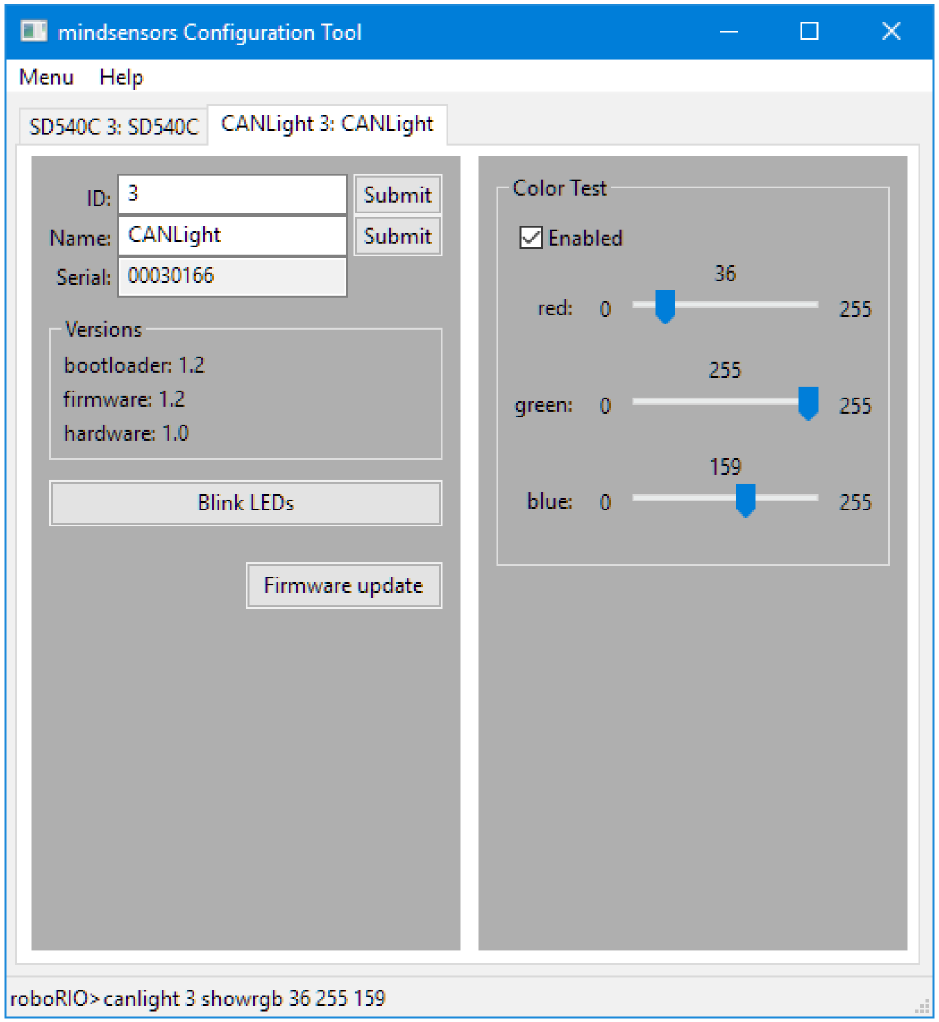
ID: 3
Name: CANLight
Serial: 00030166

Versions
bootloader: 1.2
firmware: 1.2
hardware: 1.0

☐ Enabled

red: 0 — 0 — 255
green: 0 — 0 — 255
blue: 0 — 0 — 255

```
roboRIO> canlight 3 versions {"bootloader":"1.2","firmware":"1.2","hardware":"1.0"}
```



ID: 3
Name: CANLight
Serial: 00030166

Versions
bootloader: 1.2
firmware: 1.2
hardware: 1.0

☑ Enabled

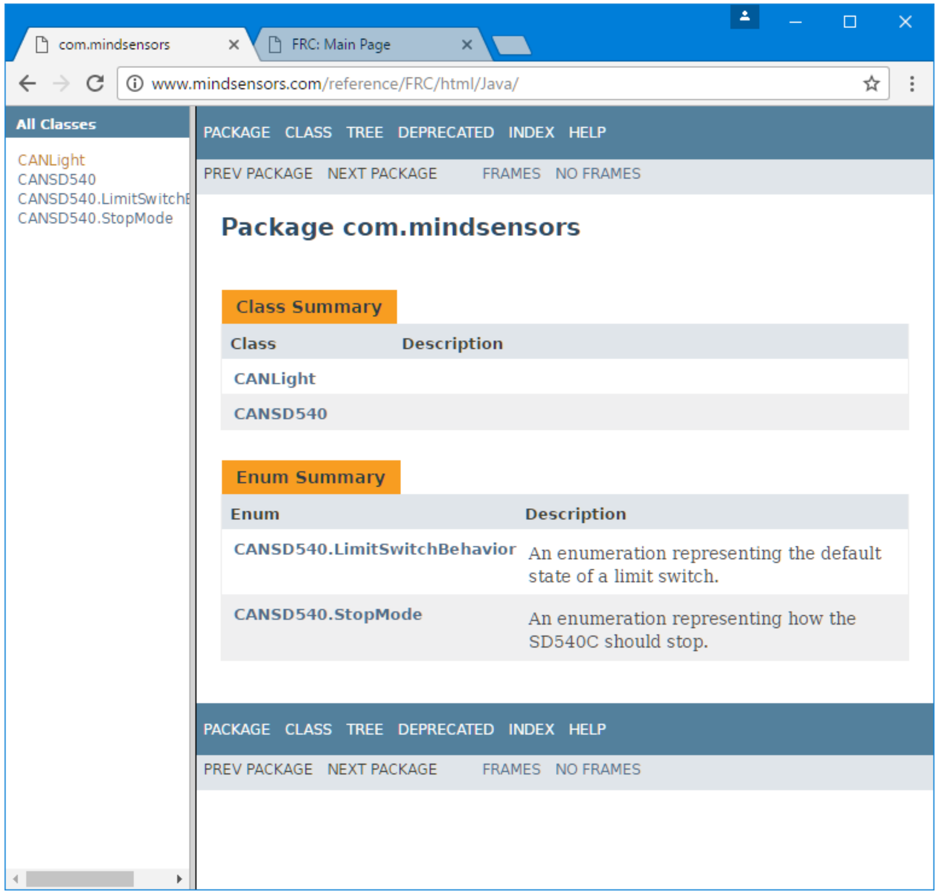
red: 0 — 36 — 255
green: 0 — 255 — 255
blue: 0 — 159 — 255

```
roboRIO> canlight 3 showrgb 36 255 159
```

How to Use the Documentation

- Documentation is an extremely useful tool (as you may know!)
- Let's say you wanted to know about *WriteRegister*. What does it do? What arguments does it take? Does it return anything?

Documentation: Home Page


com.mindsensors | FRC: Main Page

www.mindsensors.com/reference/FRC/html/Java/

All Classes

CANLight
CANSD540
CANSD540.LimitSwitchE
CANSD540.StopMode

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package com.mindsensors

Class Summary

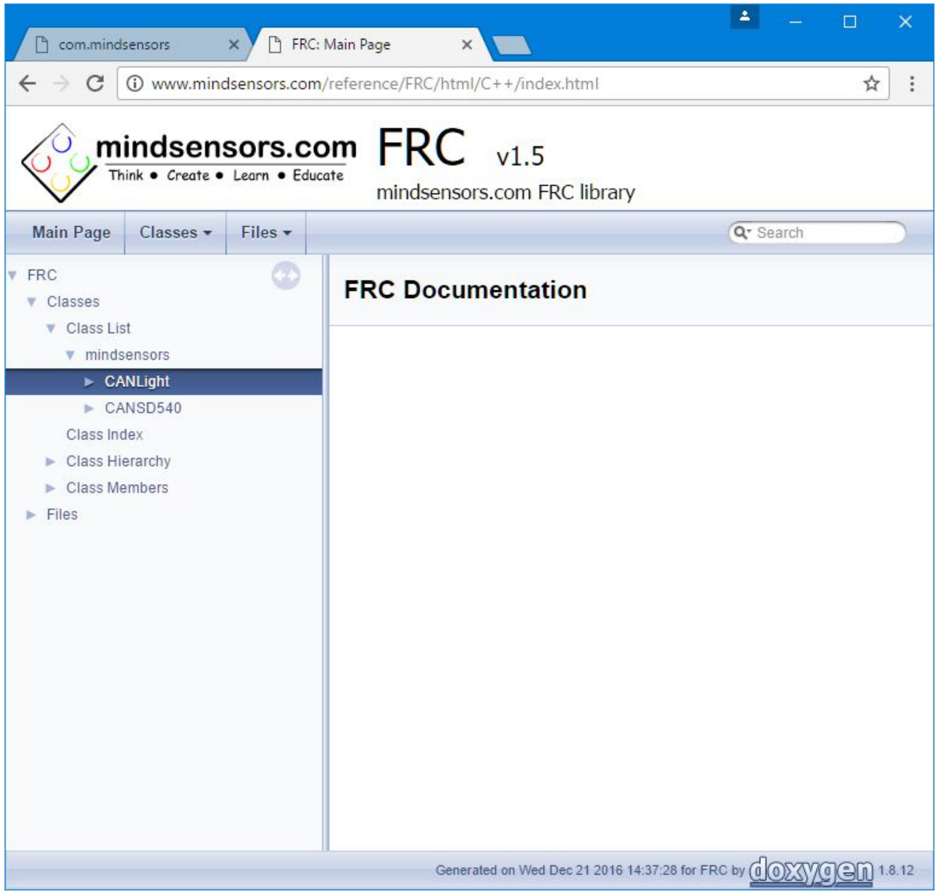
Class	Description
CANLight	
CANSD540	

Enum Summary

Enum	Description
CANSD540.LimitSwitchBehavior	An enumeration representing the default state of a limit switch.
CANSD540.StopMode	An enumeration representing how the SD540C should stop.

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES


com.mindsensors | FRC: Main Page

www.mindsensors.com/reference/FRC/html/C++/index.html

mindsensors.com Think • Create • Learn • Educate FRC v1.5

mindsensors.com FRC library

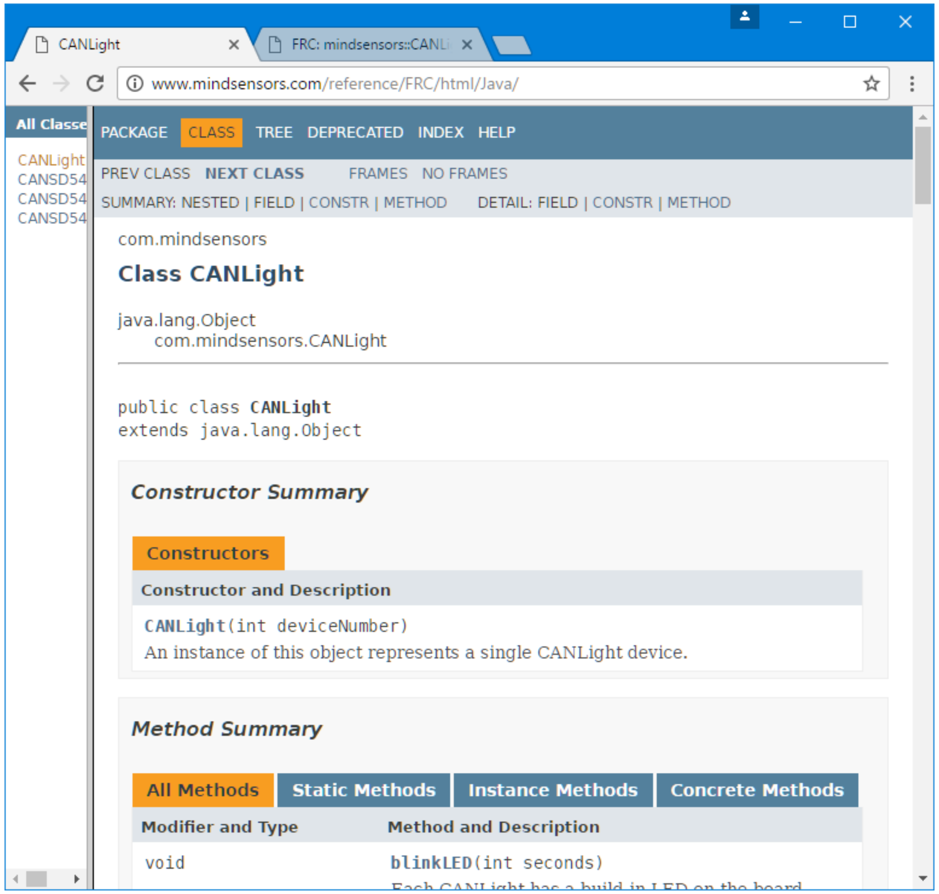
Main Page | Classes | Files | Search

- FRC
  - Classes
    - Class List
      - mindsensors
        - CANLight
        - CANSD540
    - Class Index
    - Class Hierarchy
    - Class Members
  - Files

FRC Documentation

Generated on Wed Dec 21 2016 14:37:28 for FRC by doxygen 1.8.12

Documentation: Class Page


CANLight
FRC: mindsensors::CANL
www.mindsensors.com/reference/FRC/html/Java/
All Classe
CANLight
CANSD54
CANSD54
CANSD54
PACKAGE CLASS TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.mindsensors

Class CANLight

java.lang.Object
    com.mindsensors.CANLight

```
public class CANLight
extends java.lang.Object
```

Constructor Summary

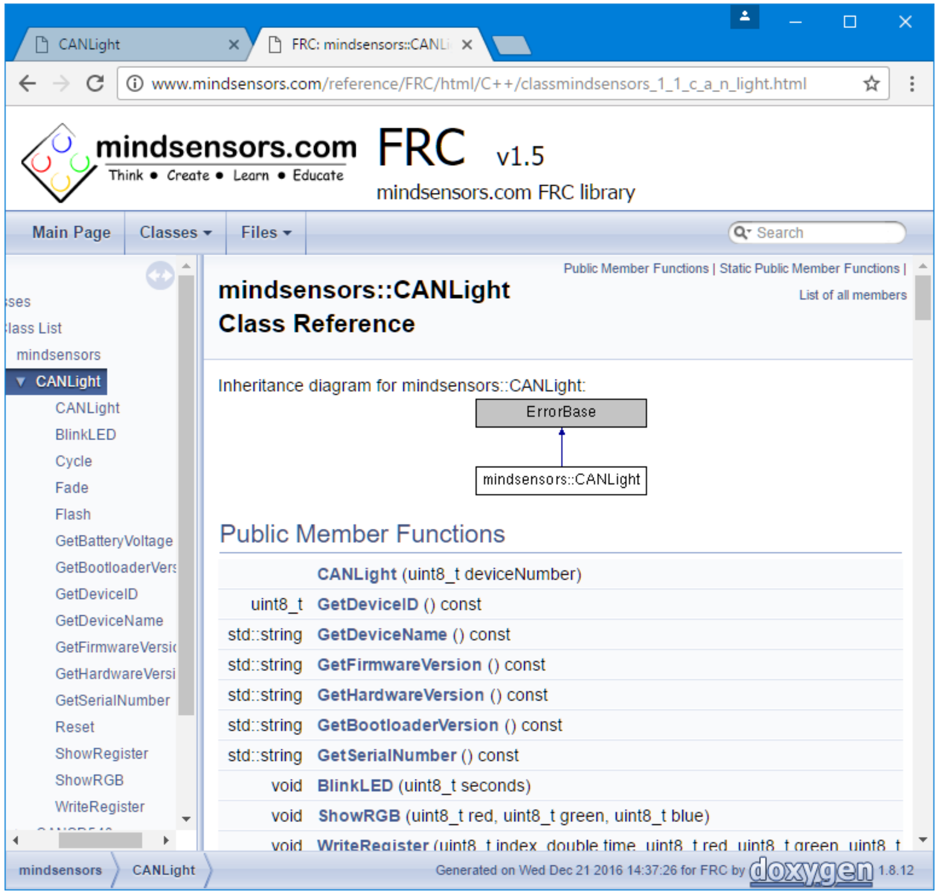
Constructors

Constructor and Description
`CANLight(int deviceNumber)` An instance of this object represents a single CANLight device.

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	`blinkLED(int seconds)` Each CANLight has a build in LED on the board



mindsensors::CANLight Class Reference

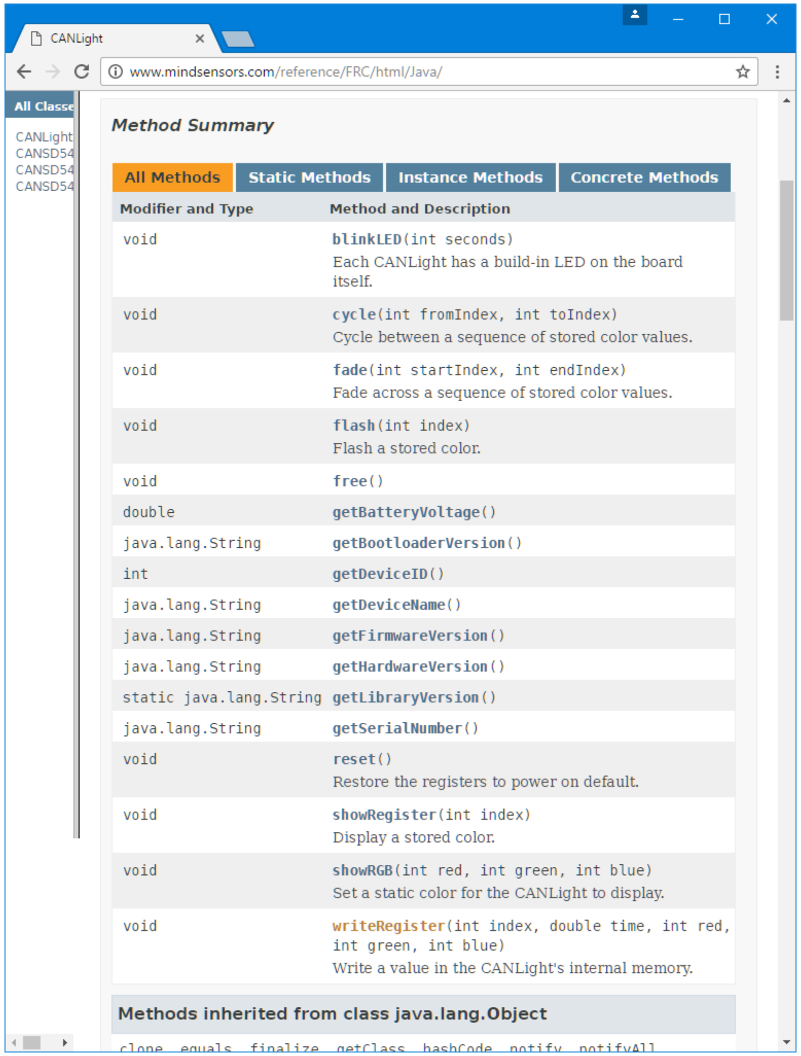
Inheritance diagram for mindsensors::CANLight:

Public Member Functions

	CANLight (uint8_t deviceNumber)
uint8_t	**GetDeviceID** () const
std::string	**GetDeviceName** () const
std::string	**GetFirmwareVersion** () const
std::string	**GetHardwareVersion** () const
std::string	**GetBootloaderVersion** () const
std::string	**GetSerialNumber** () const
void	**BlinkLED** (uint8_t seconds)
void	**ShowRGB** (uint8_t red, uint8_t green, uint8_t blue)
void	**WriteRegister** (uint8_t index, double time, uint8_t red, uint8_t green, uint8_t

Generated on Wed Dec 21 2016 14:37:26 for FRC by doxygen 1.8.12

Documentation: Method List



CANLight

www.mindsensors.com/reference/FRC/html/Java/

All Classe

CANLight
CANSD54
CANSD54
CANSD54

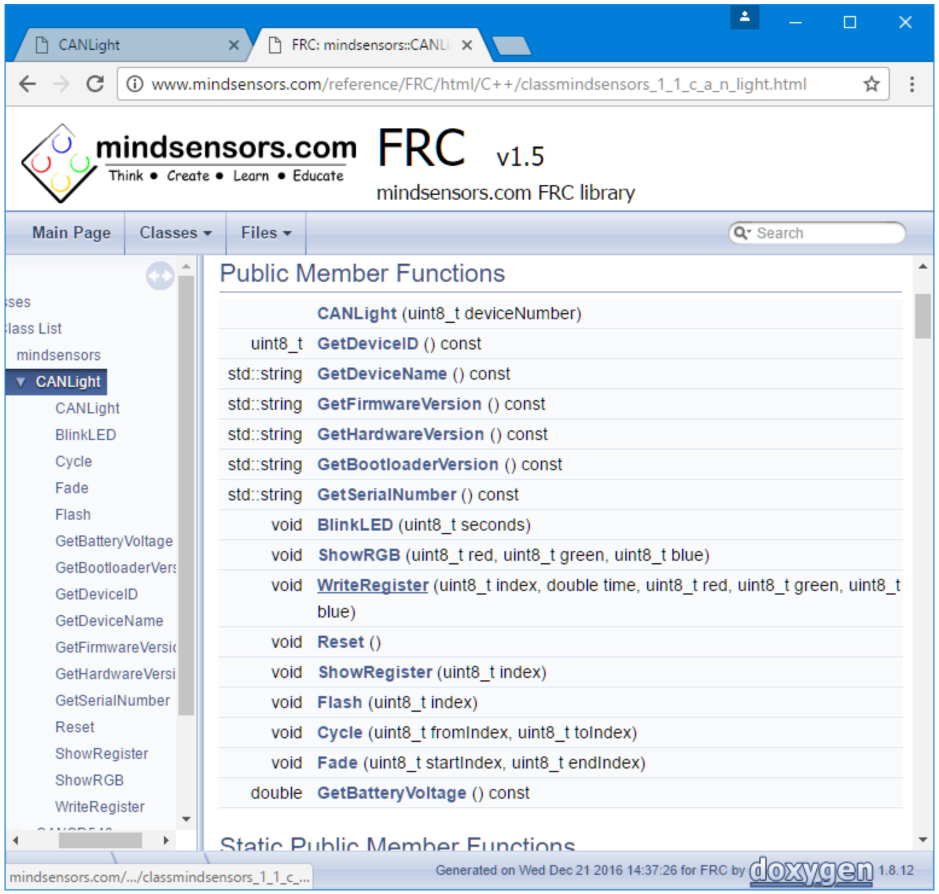
Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	`blinkLED(int seconds)` Each CANLight has a build-in LED on the board itself.
void	`cycle(int fromIndex, int toIndex)` Cycle between a sequence of stored color values.
void	`fade(int startIndex, int endIndex)` Fade across a sequence of stored color values.
void	`flash(int index)` Flash a stored color.
void	`free()`
double	`getBatteryVoltage()`
java.lang.String	`getBootloaderVersion()`
int	`getDeviceID()`
java.lang.String	`getDeviceName()`
java.lang.String	`getFirmwareVersion()`
java.lang.String	`getHardwareVersion()`
static java.lang.String	`getLibraryVersion()`
java.lang.String	`getSerialNumber()`
void	`reset()` Restore the registers to power on default.
void	`showRegister(int index)` Display a stored color.
void	`showRGB(int red, int green, int blue)` Set a static color for the CANLight to display.
void	`writeRegister(int index, double time, int red, int green, int blue)` Write a value in the CANLight's internal memory.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll



CANLight | FRC: mindsensors::CANLi

www.mindsensors.com/reference/FRC/html/C++/classmindsensors_1_1_c_a_n_light.html

mindsensors.com Think • Create • Learn • Educate

FRC v1.5

mindsensors.com FRC library

Main Page | Classes | Files | Search

ses
lass List
mindsensors
CANLight
CANLight
BlinkLED
Cycle
Fade
Flash
GetBatteryVoltage
GetBootloaderVers
GetDeviceID
GetDeviceName
GetFirmwareVersi
GetHardwareVersi
GetSerialNumber
Reset
ShowRegister
ShowRGB
WriteRegister

Public Member Functions

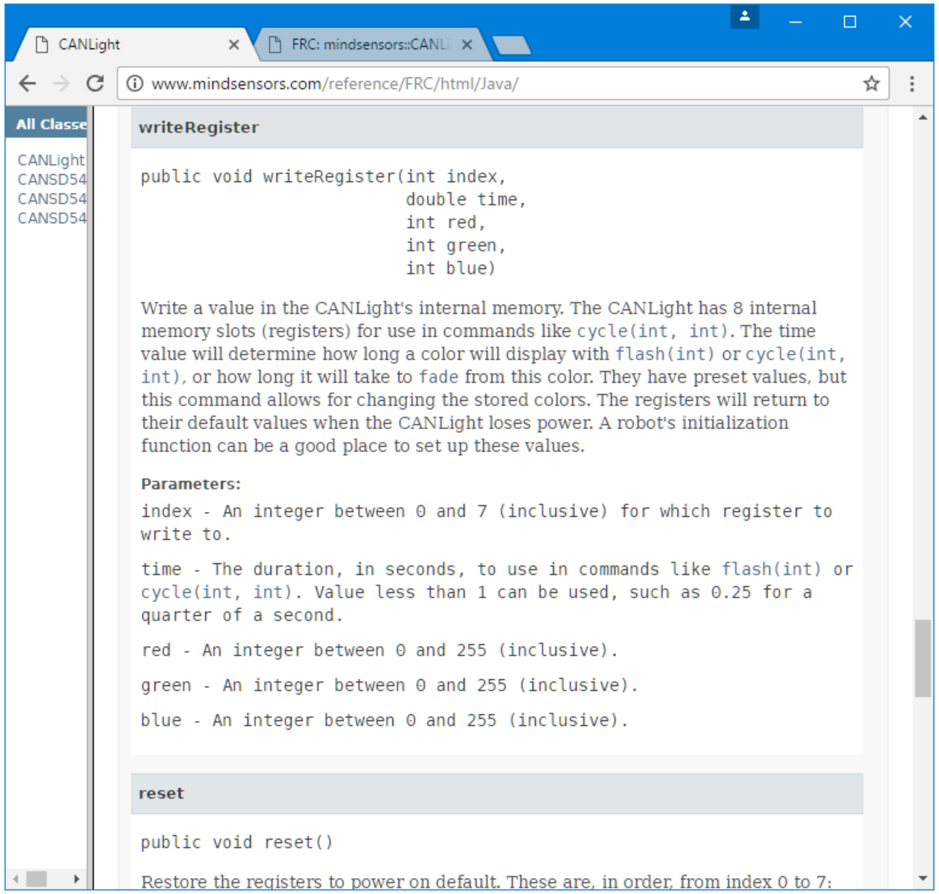
	CANLight (uint8_t deviceNumber)
uint8_t	**GetDeviceID** () const
std::string	**GetDeviceName** () const
std::string	**GetFirmwareVersion** () const
std::string	**GetHardwareVersion** () const
std::string	**GetBootloaderVersion** () const
std::string	**GetSerialNumber** () const
void	**BlinkLED** (uint8_t seconds)
void	**ShowRGB** (uint8_t red, uint8_t green, uint8_t blue)
void	**WriteRegister** (uint8_t index, double time, uint8_t red, uint8_t green, uint8_t blue)
void	**Reset** ()
void	**ShowRegister** (uint8_t index)
void	**Flash** (uint8_t index)
void	**Cycle** (uint8_t fromIndex, uint8_t toIndex)
void	**Fade** (uint8_t startIndex, uint8_t endIndex)
double	**GetBatteryVoltage** () const

Static Public Member Functions

mindsensors.com/.../classmindsensors_1_1_c_...

Generated on Wed Dec 21 2016 14:37:26 for FRC by doxygen 1.8.12

Documentation: Method Detail



writeRegister

```
public void writeRegister(int index,
                          double time,
                          int red,
                          int green,
                          int blue)
```

Write a value in the CANLight's internal memory. The CANLight has 8 internal memory slots (registers) for use in commands like `cycle(int, int)`. The time value will determine how long a color will display with `flash(int)` or `cycle(int, int)`, or how long it will take to `fade` from this color. They have preset values, but this command allows for changing the stored colors. The registers will return to their default values when the CANLight loses power. A robot's initialization function can be a good place to set up these values.

Parameters:

`index` - An integer between 0 and 7 (inclusive) for which register to write to.

`time` - The duration, in seconds, to use in commands like `flash(int)` or `cycle(int, int)`. Value less than 1 can be used, such as 0.25 for a quarter of a second.

`red` - An integer between 0 and 255 (inclusive).

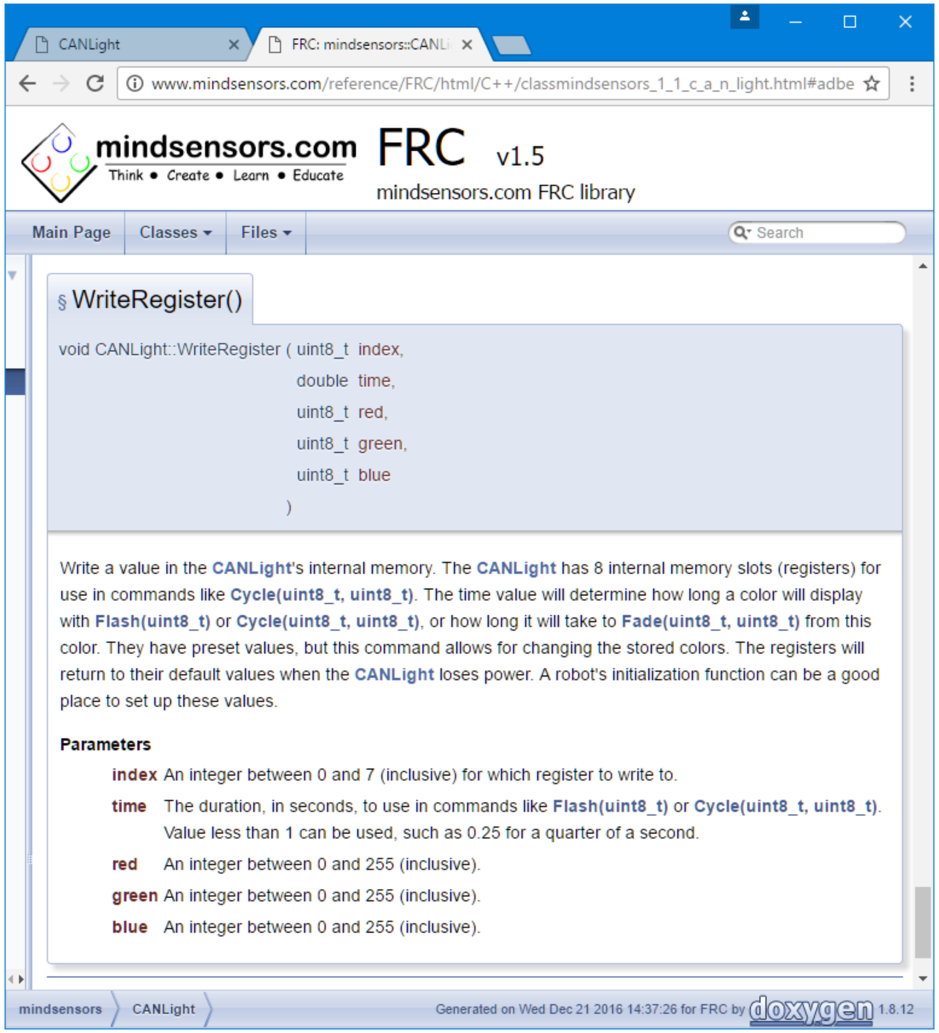
`green` - An integer between 0 and 255 (inclusive).

`blue` - An integer between 0 and 255 (inclusive).

reset

```
public void reset()
```

Restore the registers to power on default. These are, in order, from index 0 to 7:



CANLight
FRC: mindsensors::CANLi
www.mindsensors.com/reference/FRC/html/C++/classmindsensors_1_1_c_a_n_light.html#adbe
mindsensors.com
Think • Create • Learn • Educate
FRC v1.5
mindsensors.com FRC library
Main Page
Classes
Files
Search

§ WriteRegister()

```
void CANLight::WriteRegister ( uint8_t  index,
                               double   time,
                               uint8_t  red,
                               uint8_t  green,
                               uint8_t  blue
                             )
```

Write a value in the CANLight's internal memory. The CANLight has 8 internal memory slots (registers) for use in commands like **Cycle(uint8_t, uint8_t)**. The time value will determine how long a color will display with **Flash(uint8_t)** or **Cycle(uint8_t, uint8_t)**, or how long it will take to **Fade(uint8_t, uint8_t)** from this color. They have preset values, but this command allows for changing the stored colors. The registers will return to their default values when the CANLight loses power. A robot's initialization function can be a good place to set up these values.

Parameters

index	An integer between 0 and 7 (inclusive) for which register to write to.
time	The duration, in seconds, to use in commands like **Flash(uint8_t)** or **Cycle(uint8_t, uint8_t)**. Value less than 1 can be used, such as 0.25 for a quarter of a second.
red	An integer between 0 and 255 (inclusive).
green	An integer between 0 and 255 (inclusive).
blue	An integer between 0 and 255 (inclusive).

mindsensors 〉 CANLight 〉 Generated on Wed Dec 21 2016 14:37:26 for FRC by doxygen 1.8.12

Precautions

- The CANLight will draw a lot of attention to your robot
  - People like bright, colorful lights
- Please be tasteful with the speeds, intensity, and contrasting colors you use
  - Your robot represents your team
  - 255 is very bright! It can have a lot more impact if you use lower values normally, and momentarily use very bright colors when something special happens (score a goal, sudden impact, etc.)

Resources

- mindsensors.com FRC library blog post:
  http://www.mindsensors.com/blog/how-to/how-to-use-sd540c-and-canlight-with-roborio
- mindsensors.com FRC library:
  http://www.mindsensors.com/largefiles/FIRST/mindsensors.zip
- mindsensors Configuration Tool blog post:
  http://www.mindsensors.com/blog/how-to/using-the-mindsensors-configuration-tool
- mindsensors Configuration Tool:
  http://www.mindsensors.com/largefiles/FIRST/mindsensorsConfigurationTool.zip
- mindsensors.com FRC library documentation in Java:
  http://www.mindsensors.com/reference/FRC/html/Java/
- mindsensors.com FRC library documentation in C++:
  http://www.mindsensors.com/reference/FRC/html/C++/
- WPI tutorials:
  https://wpilib.screenstepslive.com/s/4485