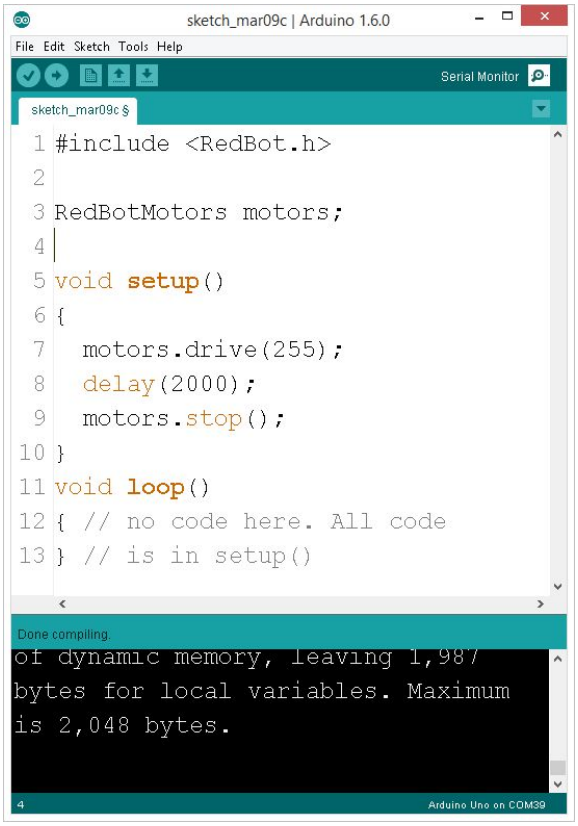
Name: _____ Date: _____

RedBot Library
Quick Reference Guide



Introduction

We have written our own library to simplify the programming and interface to the RedBot motors, encoders, sensors, and other peripherals. This is a quick summary / overview of the RedBot Library, classes, methods, and variables.

The RedBot library has a number of pre-written functions and commands that make controlling the robot easier. To use the RedBot Library, download and install the library into Arduino. You can download the latest version of the library at: http://sfe.io/RedBotLib.

Unzip the contents of this folder into the **libraries** folder inside the Arduino sketchbook (documents folder).

When using the RedBot Library, you have to "include" it into your code. Do this using this single line of code at the top. Pay attention to the case, and notice that there is not a semicolon at the end of this line.

```
#include <RedBot.h>
```

Library Reference

You may instantiate objects using any of the classes described below. Many of these are used throughout the tutorials around the RedBot. Feel free to play around with these features on your device.

RedBotMotors class

`RedBotMotors motors;` – this creates an instance of the RedBotMotors class. This can only be instantiated once in your code. This allows you to control the motors with simple methods such as:

- `.drive(motorPower)` – this method drives the right side CW and the left side CCW for positive values of motorPower (i.e. drives the RedBot forward), and it does the reverse for negative values of motorPower.
- `.pivot(motorPower)` – this method spins the entire RedBot CW for positive values of motorPower and CCW for negative values of motorPower.

- `.coast()` – stops the motors and allows the RedBot to coast to a stop.
- `.brake()` – applies the brakes using the H-Bridge by shorting out both of the motors. Forces the motors to come to an abrupt stop.
- `.leftMotor(motorPower)` – controls the leftMotor independantly. Positive values of motorPower spin the motor CW, and negative values spin the motor CCW.
- `.leftBrake(motorPower)` – applies the brakes the leftMotor.
- `.leftCoast(motorPower)` – stops the leftMotor allowing it to coast to a stop.
- `.rightMotor(motorPower)` – controls the rightMotor independantly. Positive values of motorPower spin the motor CW, and negative values spin the motor CCW.
- `.rightBrake(motorPower)` – applies the brakes the rightMotor.
- `.rightCoast(motorPower)` – stops the rightMotor allowing it to coast to a stop.

Advanced

If you wish to control the motors independently, the following are the control pins for the left and right motors:

```
leftMotor_controlPin1 = 2
leftMotor_controlPin2 = 4
leftMotor_motorPwrPin = 5

rightMotor_controlPin1 = 7
rightMotor_controlPin2 = 8
rightMotor_motorPwrPin = 6
```

The RedBot uses the Toshiba TB6612FNG H-Bridge Motor Driver. This is a common H-Bridge Motor Driver used in robotics. Set the control pins either HIGH / LOW or LOW / HIGH to set the direction of the motor. Using a PWM signal, set the speed of the motor with the motorPwrPin.

RedBotEncoder class

`RedBotEncoder encoder(leftEncoder, rightEncoder);` – this creates a RedBotEncoder object with the left encoder connected to pin `leftEncoder` and the right encoder connected to pin `rightEncoder`. This can only be instantiated once in your code.

- `.clearEnc(LEFT\RIGHT\BOTH)` – clears the counter variable for either the `LEFT`, `RIGHT`, or `BOTH` encoders. The labels `LEFT`, `RIGHT`, and `BOTH` are specific types defined in this class.

- `.getTicks(LEFT\RIGHT)` – returns a long integer value representing the number of encoder ticks (counts) for either the `LEFT` or the `RIGHT` encoder.

RedBotButton class

`RedBotButton button();` – Creates an instance of the RedBotButton class on the RedBot. Because the button is hardwired on the RedBot board to pin 12, this initialization is handled inside the library code. This can only be instantiated once in your code.

- `.read()` – returns a boolean value representing the status of the button.

RedBotSensor class

`RedBotSensor sensorA(pinNum);` – this creates a RedBotSensor object connected to the port (pin) on the RedBot Mainboard. This is primarily used for the line-following sensors, but can be used with any analog sensor. This class can only be instantiated for as many sensors as you have on your RedBot.

- `.read()` – returns an integer value scales from 0 to 255 for the analog voltage read by the sensor. 0 represents 0V and 255 represents 5V.
- `.setBGLevel()` – reads the value of the sensor in it's 'nominal' position. This value is stored as the 'background' level.
- `.setDetectLevel()` – reads the value of the sensor in it's 'detect' position. This value is stored as the threshold for a 'detect' level.
- `.check()` – returns a boolean value that TRUE if the measured sensor value is greater than ¼ of the difference between the background and the detect levels. Note: This method only works if you have set both the BGLevel and the DetectLevel.

RedBotBumper class

`RedBotBumper bumperA(pinNum);` – Creates an instance of RedBotBumper object connected to the port (pin) on the RedBot Mainboard. While the RedBotBumper is designed for use with the whisker / bumper switches on the RedBot, it can be used with any digital sensor or switch. This class can only be instantiated for as many digital sensors (bumpers) as you have on your RedBot.

- `.read()` – returns a boolean value for the state of the bumper. It returns TRUE when the bumper switch is closed or connected to GND.

RedBotAccel class

`RedBotAccel accel();` – Creates an instance of the RedBotAccel object. The RedBot accelerometer uses I2C for communication. Because of this, it needs to be connected to A4/A5 on the RedBot Mainboard. This can only be instantiated once in your code.

- `.read()` – this reads the current values of the accelerometer and stores it into local variables in the library. It does not return any values.
- `.x` – is the raw X-axis accelerometer value read using the `.read()` method. In general, these values vary from -16000 to +16000. The X-axis is aligned with the FWD/REV direction of the RedBot.
- `.y` – is the raw Y-axis accelerometer value read using the `.read()` method. In general, these values vary from -16000 to +16000. The Y-axis is aligned with the lateral or RIGHT/LEFT direction of the RedBot.
- `.z` – is the raw Z-axis accelerometer value read using the `.read()` method. In general, these values vary from -16000 to +16000. The Y-axis is aligned with the UP/DOWN direction of the RedBot.
- `.angleXZ` – is a floating point value for the calculated angle between the X and Z axes of the accelerometer. It calculates the arc tangent between the raw X and raw Z values. On the RedBot, this is the forward / backward tilt.
- `.angleYZ` – is a floating point value for the calculated angle between the Y and Z axes of the accelerometer. It calculates the arc tangent between the raw Y and raw Z values. On the RedBot, this is the side to side tilt.
- `.angleXY` – is a floating point value for the calculated angle between the X and Y axes of the accelerometer. It calculates the arc tangent between the raw X and raw Y values.

RedBotSoftwareSerial class

`RedBotSoftwareSerial xBeeRadio();` – Creates an instance of the RedBotSoftwareSerial object. This library uses a lot of code from the standard Arduino `SoftwareSerial` library. The RedBot Mainboard uses pins A0 (14) and A1 (15) for TX and RX when switched to SW_SERIAL.

- `.begin(baudRate)` – opens up the serial communication on the SW_SERIAL TX and RX lines of the RedBot Mainboard at the `baudRate`. Acceptable baud rates include: 300, 1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200.
- `.write()` – writes a single bye of data to the software serial port.
- `.read()` – returns a single byte from the software serial port.
- `.available()` – returns an integer value representing the number of bytes (characters) available for reading from a software serial port.