1 Introduction

1.1 Overview

The RobotOpen Arduino Library provides an easy to use abstraction layer to quickly begin programming and controlling your robots with RobotOpen. The 1.1.x release is currently compatible with the RobotOpen Control Shield for Arduino coupled with the boards listed below.

1.2 Supported Hardware

- Ethernet Arduino
- Arduino Uno + Ethernet Shield
- Arduino Duemilanove + Ethernet Shield
- Sparkfun Ethernet Pro

1.3 Recommended Hardware

- Windows Based PC with Java 6+
- Logitech Dual Action USB Game Pad
- Ethernet Arduino + RobotOpen Control Shield

1.4 Installation

Before installing the RobotOpen library, first verify that you are running the latest version of the Arduino IDE. The latest version of the RobotOpen Arduino library can be found at www.robotopen.biz. Download and extract the zip file. You should be left with a folder named "RobotOpen". You can then place this folder into your Arduino installation directory. On Windows this will be located under Arduino/libraries. On OS X you will need to right click the Arduino.app file and select "Show Package Contents". Navigate to Contents/Resources/Java/libraries and place the RobotOpen folder in this directory. If you are upgrading your version of the RobotOpen library you can safely delete the "RobotOpen" directory before copying in the new version.

1.5 Driver Station App

The Driver Station app is a simple Java application to allow control of your RobotOpen powered robot. This can be obtained from www.robotopen.biz. Please view the getting started video listed on the site to learn more.

1.6 Examples

Now that you have the RobotOpen library loaded up, simply open the Arduino IDE and select File » Examples » RobotOpen » TankDrive. This is the simplest example bunded into the RobotOpen library that shows how you should model your RobotOpen projects. For this example, all of the code is provided to get a basic robot up and running in a Tank Drive configuration. By default, the Logitech Dual Action USB Game Pad will work out of the box.

2 RobotOpen Class

The RobotOpen class is the core of the RobotOpen library. It is important to note that the inner workings of RobotOpen use what are called "bundles" to send and receive data between the driver station and robot controller. Every bundle is identified by a bundle ID that is a single character. Typically 0-9, a-z, or A-Z. When accessing the low level data in RobotOpen, you will use these bundle IDs.

2.1 RobotOpen::begin()

Returns: void

This function initializes communications and configures RobotOpen. This should only be called once from the setup() function in the Arduino environment.

2.2 RobotOpen::pollDS()

Returns: void

This function processes any data that has been received from the driver station and updates all joystick objects. It should be called at the beginning of every program loop.

2.3 RobotOpen::outgoingDS()

Returns: void

This function notifies RobotOpen that all of the robot code has finished executing and any data waiting to be sent back to the driver station should happen now. This should be called at the end of every program loop.

2.4 RobotOpen::setPWM(int pwmChannel, int value)

int pwmChannel: The PWM output to set. See the Digital Sidecar IO Constants near the end of this document for possible values. Example: SIDECAR_PWM7.
int value: The PWM value to set. Use a joystick makePWM function here or manually pass a value between 0-255.
Returns: void

This function sets a motor or PWM value on a specific PWM output of the digital sidecar.

2.5 RobotOpen::setRelay(int relayChannel, int value)

int relayChannel: The relay output to set. See the Constants section near the end of this document for possible values. Example: RELAY_3.
int value: The relay value to set. Possible values here are either ON or OFF.
Returns: void

This function sets a motor or PWM value on a specific PWM output of the digital sidecar.

2.6 RobotOpen::enabled()

Returns: boolean

This will return to you a boolean of the current state of the robot. True corresponds to enabled. False corresponds to disabled.

2.7 RobotOpen::publishAnalog(int pin, unsigned char bundleID)

int pin: The analog pin on the RobotOpen Shield that you would like to send back to the DS. The Analog Input Constants listed below can be passed in for this value.
unsigned char bundleID: A unique single character that identifies this bundle. This should be passed in single quotes. Typical values include 0-9, a-z, and A-Z.
Returns: void

This method measures the voltage of the specified analog pin and sends it back to the driver station as a value between 0-1023. The bundleID is simply a method to identify the data "bundle" on your driver station.

2.8 RobotOpen::publishDigital(int pin, unsigned char bundleID)

int pin: The digital pin on the RobotOpen Shield that you would like to send back to the DS. The Digital Sidecar IO Constants listed below can be passed in for this value.
unsigned char bundleID: A unique single character that identifies this bundle. This should be passed in single quotes. Typical values include 0-9, a-z, and A-Z.
Returns: void

This method measures a digital sidecar pin state (0vdc or 5vdc) and sends it back to the driver station as either a 0 (0vdc signal) or 255 (5vdc signal). The bundleID is simply a method to identify the data "bundle" on your driver station.

2.9 RobotOpen::publishByte(unsigned char value, unsigned char bundleID)

unsigned char value: The single byte (8 bit) value that you would like to send back to the DS.
unsigned char bundleID: A unique single character that identifies this bundle. This should be passed in single quotes. Typical values include 0-9, a-z, and A-Z.
Returns: void

This method sends a single byte back to the driver station. The bundleID is simply a method to identify the data "bundle" on your driver station.

2.10 RobotOpen::publishInt(unsigned int value, unsigned char bundleID)

unsigned int value: The int (16 bit) value that you would like to send back to the DS.
unsigned char bundleID: A unique single character that identifies this bundle. This should be passed in single quotes. Typical values include 0-9, a-z, and A-Z.
Returns: void

This method sends a single int back to the driver station. The bundleID is simply a method to identify the data "bundle" on your driver station.

2.11 RobotOpen::publishLong(long value, unsigned char bundleID)

long value: The long (32 bit) value that you would like to send back to the DS.
unsigned char bundleID: A unique single character that identifies this bundle. This should be passed in single quotes. Typical values include 0-9, a-z, and A-Z.
Returns: void

This method sends a single long back to the driver station. The bundleID is simply a method to identify the data "bundle" on your driver station.

2.12 RobotOpen::getBundleSize(unsigned char bundleID)

Returns: int

All data from the driver station is sent in bundles. Normally you will only deal with joystick values coming from your driver station. Each joystick should be in its own bundle. Joystick 1 is typically in bundle '0' and Joystick 2 is typically in bundle '1'. This function allows you to get the size (number of bytes) of a particular bundle that you are receiving from the DS. If the bundle cannot be found, -1 is returned.

2.13 RobotOpen::getComponent(unsigned char bundleID, int componentIndex)

Returns: int

All data from the driver station is sent in bundles. Normally you will only deal with joystick values coming from your driver station. Each joystick should be in its own bundle. Joystick 1 is typically in bundle '0' and Joystick 2 is typically in bundle '1'. This function allows you to get the value from a specific joystick component or byte in your joystick bundle. componentIndex allows you to specify a byte index between 0 (first component) and the number of components in the bundle - 1. Valid values range between 0-255. An error will be returned as -1.

3 USBJoystick Class

The USBJoystick class allows creation of usb objects from bundle IDs to act as an abstraction layer.

Example: USBJoystick usb1('0');
This creates a USBJoystick object that can now be accessed as usb1.function() using the values in bundle '0'.

3.1 USBJoystick(unsigned char bundleID)

The constructor for creating a new USBJoystick object. bundleID specifies the bundle to create the USBJoystick object from. Joystick 1 is typically in bundle '0' and Joystick 2 is typically in bundle '1'.

3.2 joystick.getIndex(int index)

int index: The component index between 0 (first component) and the number of components in the bundle - 1.
Returns: int

This function allows you to grab the raw value of any component in this USBJoystick. Valid values range between 0-255. An error will be returned as -1.

3.3 joystick.makePWM(int index, char mode)

int index: The joystick axis to create the PWM value from. See Logitech Gamepad Constants below. A raw joystick index can also be passed in here.
char mode: How to generate the PWM value (INVERT or NORMAL)
Returns: int

This function will return a PWM value between 0-255 if the joystick axis/component you pass is valid or 127 (neutral) if there is an error. You will normally use this function chained with a setPWM function such as:
RobotOpen::setPWM(SIDECAR_PWM9, usb1.makePWM(ANALOG_LEFTY, NORMAL));

3.4 joystick.getBtn(int index, char mode)

int index: The joystick button to read. See Logitech Gamepad Constants below. A raw joystick index can also be passed in here.
char mode: How to generate the returned boolean (INVERT or NORMAL)
Returns: boolean

This function will return true if the specified button is being pressed or false if it is not being pressed (if the mode is NORMAL). The opposite holds true if the mode is INVERT.

3.5 joystick.getDpad(unsigned char compare, char mode)

unsigned char compare: The position of the D-Pad to check against (e.g. UP_RIGHT).
char mode: How to generate the returned boolean (INVERT or NORMAL)
Returns: boolean

This function will return true if the specified d-pad direction is being pressed or false if it is not being pressed (if the mode is NORMAL). The opposite holds true if the mode is INVERT.

4 Constants

4.1 Analog Inputs

- ANALOG0 - 0x00
- ANALOG1 - 0x01
- ANALOG2 - 0x02
- ANALOG3 - 0x03
- ANALOG4 - 0x04
- ANALOG5 - 0x05

4.2 Digital Sidecar IO

- SIDECAR_DIGITAL1 - 0x07
- SIDECAR_DIGITAL2 - 0x06
- SIDECAR_DIGITAL3 - 0x05
- SIDECAR_DIGITAL4 - 0x04
- SIDECAR_DIGITAL5 - 0x03
- SIDECAR_DIGITAL6 - 0x02
- SIDECAR_DIGITAL7 - 0x09
- SIDECAR_DIGITAL8 - 0x08
- SIDECAR_PWM1 - 0x01
- SIDECAR_PWM2 - 0x02
- SIDECAR_PWM3 - 0x03
- SIDECAR_PWM4 - 0x04
- SIDECAR_PWM5 - 0x05
- SIDECAR_PWM6 - 0x06
- SIDECAR_PWM7 - 0x07
- SIDECAR_PWM8 - 0x08
- SIDECAR_PWM9 - 0x09
- SIDECAR_PWM10 - 0x0a

4.3 Relay Outputs

- RELAY_1 - 0x01
- RELAY_2 - 0x02
- RELAY_3 - 0x03
- RELAY_4 - 0x04
- RELAY_5 - 0x05
- RELAY_6 - 0x06
- RELAY_7 - 0x07
- RELAY_8 - 0x08
- RELAY_9 - 0x09
- RELAY_10 - 0x0a

4.4 Joystick Identifiers

- ANALOG_LEFTX - 0x00
- ANALOG_LEFTY - 0x01
- ANALOG_RIGHTX - 0x02
- ANALOG_RIGHTY - 0x03
- LEFT_ANALOG_BTN_DPAD - 0x04
- RIGHT_ANALOG_BTN_DPAD - 0x05
- DPAD - 0x06
- BTN1 - 0x07
- BTN2 - 0x08
- BTN3 - 0x09
- BTN4 - 0x0a
- BTN5 - 0x0b
- BTN6 - 0x0c
- BTN7 - 0x0d
- BTN8 - 0x0e
- BTN9 - 0x0f
- BTN10 - 0x10

4.5 D-Pad Positions

- UP - 0x3F
- UP_LEFT -0x1F
- UP_RIGHT - 0x5F
- DOWN - 0xBF
- DOWN_LEFT - 0xDF
- DOWN_RIGHT - 0x9F
- LEFT - 0xFF
- RIGHT - 0x7F

4.6 Joystick Modifiers

- NORMAL - 0x00
- INVERT - 0x01