# KIPR Link Manual

**Version 0.1**

KIPR makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein. KIPR products are not intended for use in medical, life saving or life sustaining applications. KIPR retains the right to make changes to these specifications at any time, without notice.

BOTBALL®, BYO-BOT®, BOTGUY, and the BOTGUY design and character are trademarks and/or service marks of KISS Institute for Practical Robotics and may not be used without express written permission.

LEGO, iRobot, and iRobot Create are registered marks of their respective owners.

The KISS Institute is a 501c3 nonprofit organization. Our mission is to improve the public's understanding of science, technology, engineering, and math; develop the skills, character, and aspirations of students; and contribute to the enrichment of our school systems, communities, and the nation.

# Contents

# 1. KIPR Link

## About the KIPR Link

The KIPR Link is a Linux-based robot controller designed by the KISS Institute for Practical Robotics for use with the KISS IDE.

Featuring significant hardware and usability improvements over its predecessor, the KIPR Link is both a beginner-friendly choice for newcomers to robotics, and a powerful, feature-rich device that will appeal to experts.

## KIPR Link Basic Features

- GNU/Linux based operating system
- Open-source robot control software
- Integrated color vision system
- 800MHz ARMv5te processor
- Spartan-6 FPGA
- Integrated battery and charge system
- Internal speaker
- 320 x 240 color touch screen

## Input and Output

**1** - 3 axis 10-bit accelerometer (software selectable 2/4/8g)
**8** - digital I/O ports (hardware selectable 3.3V or 5V)
**8** - 3.3V (5V tolerant) 10-bit analog input ports
**4** - servo motor ports
**4** - PID motors ports with full 10-bit back EMF and PID motor control
**1** - 3.3V (5V tolerant) TTL serial port
**2** - USB A host ports for connecting devices
**1** - Micro USB port to connect to your computer
**1** - physical button
**1** - IR emitter
**1** - IR receiver
**1** – HDMI out port

## Other Features

- All sensor inputs have software enabled pull up resistor (digital 47k, analog 15k)
- Motor current up to 1A per port
- Servo ports output 6V
- I2C interface (with additional hardware)
- Arm 7 debug port (with additional hardware)
- JTAG port
- Vcc maximum current 500mA @3.3V, 1A @5V
- 7.4V 2000mAh Lithium Polymer battery pack (2s1p) 8C max discharge
- 1GB internal micro SD for storage
- Internal 802.11 b/g wif

## Included Hardware

Each KIPR Link includes the KIPR Link, USB cable, power adapter and USB camera.

**KIPR Link**

**USB cable**

**USB Camera**

**Power Adapter**

# 2. Quick Start

## Turn On Your KIPR Link

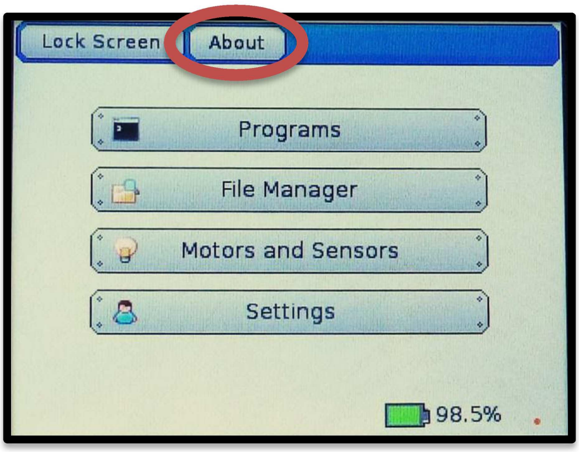
Plug you AC adapter into the wall and into the back of the KIPR Link. You will see a green charger present light and a colored charge status light. Slide the power switch to the on position to boot the KIPR Link. The KIPR Link will take 30-45 seconds to boot into the user interface. When the KIPR Link is fully booted you will see a screen similar to below.

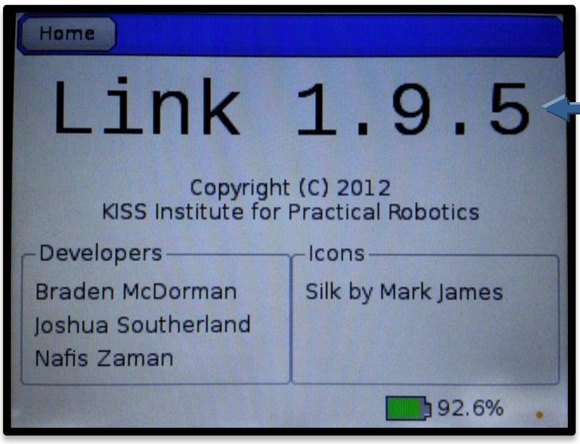
## Check the Software on Your KIPR Link

The CBC is controlled by a main software component called the Firmware. Check the KIPR Link page at http://www.kipr.org/products/link to see and download the most recent release.

To **check the version of the software currently running on your KIPR Link**, boot your KIPR Link and press the About button at the top of the home screen.

In the middle of the screen, check the firmware version. If the firmware is out of date, you can follow the instructions in the Appendices to update your KIPR Link to the latest release.





Link 1.9.5

Copyright (C) 2012
KISS Institute for Practical Robotics

Developers
Braden McDorman
Joshua Southerland
Nafis Zaman

Icons
Silk by Mark James

92.6%

This number should match the firmware number on the website.

# Install Software on Your Computer

## Mac OS X 10.5 and higher:

First, **install the most recent release of Xcode for your version of Os X**. You can find the Xcode installer on your Os X install disks, or download it from the Apple Developer Connection at http://developer.apple.com (you must register for a free account to login and download the software).

If you are running 10.7 or higher you will also need to install the command line tools. Go to the Xcode menu, then select "Preferences" and click the "Downloads" icon. Click the install button next to "Command Line Tools".

Next, double click on "KISS Platform *x.x.x*.dmg" file to mount the disk image. **Copy the KISS Platform folder in the disk image to the Applications folder on your Mac.** You need to keep the KISS-IDE application and the library folders in the same KISS-IDE folder (programs you write can be kept wherever you wish).

There is no need to install a USB driver; appropriate drivers come with Os X.

## Windows XP, Vista, and 7:

First, **connect the KIPR Link to the computer** using the included USB cable and turn on the KIPR Link.

**Double click on KISS Platform installer** (KISS Platform *x.x.x*.exe ). That will open the KISS IDE Intallation Wizard. Click Next to begin the installation process.

On the Choose Components screen, make sure that both KISS-C and the CBC driver are selected, then click Next to choose an install location. It is recommended that you install KISS IDE in the default Program Files folder**. Click Install to begin installing KISS IDE to your computer.**

When the driver gets installed you will be prompted that the driver is unsigned. This is normal. Continue the install anyway.

In Windows XP, you may be prompted to install the driver. Click next and then click next again (search for driver) and Windows XP will find and install the driver. KISS IDE will be added to your program menu. A KISS IDE shortcut will be placed on your desktop.

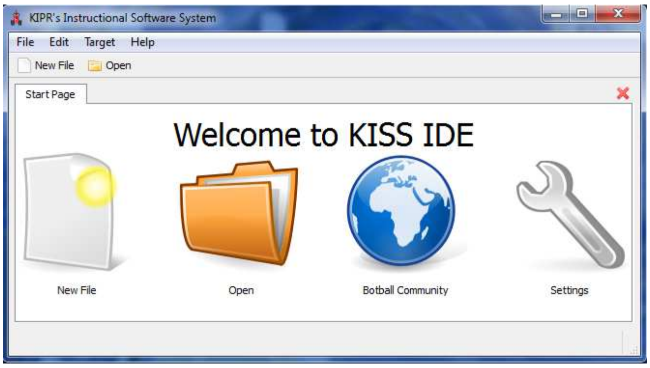
## Windows 8:

**Do Not Connect KIPR Link to Computer Until Step 11!**

1. Boot into Windows 8 normally
2. Move mouse into top right corner and click on the search icon
   a. Search for “Settings”
   b. Click settings
   c. Search for “General”
   d. Click on General Settings
3. Choose Advanced Startup
   a. Click “Restart Now”
4. Click “Troubleshoot”
5. Click “Advanced Options”
6. Click “Startup Settings”
7. Click “Restart”
8. Choose option 7
9. Log in to windows 8
10. Plug in your KIPR Link via USB cable
11. Turn on your KIPR Link
12. Install KISS IDE 4.0.0+
13. Search for “Device Manager”
    a. You should see “Gadget Serial v2.4”
    b. Right click on “Gadget Serial v2.4” and choose update driver software
    c. Then choose to search automatically
    d. A low COM port is good (anything less than 10 should work fine)

# Download a Program to Your KIPR Link

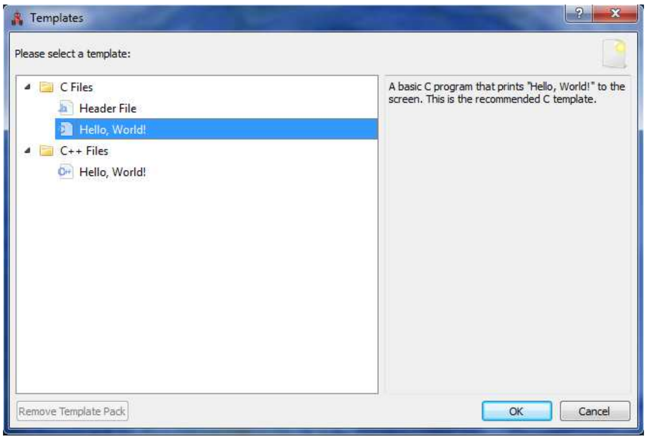
## 1. Connect the KIPR Link to your Computer



Turn on the KIPR Link on and connect it to your computer with the USB cable.

Launch KISS. When the program starts up, the Welcome Screen will appear. Click New File from this menu click to continue.

## 2. Select a Template

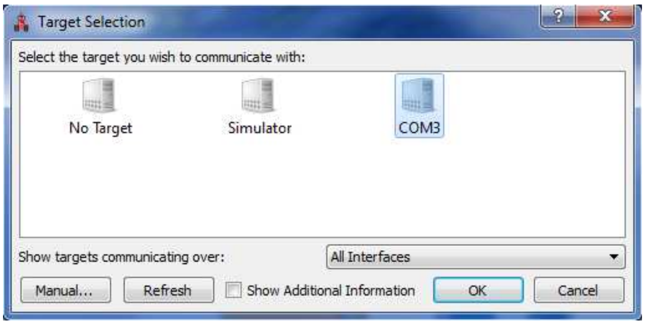


Please select a template:

A basic C program that prints "Hello, World!" to the screen. This is the recommended C template.

The Templates Window will pop up. Select “Hello, World!” under the C Files folder and then click OK to continue.

## 3. Select a Target



Select the target you wish to communicate with:

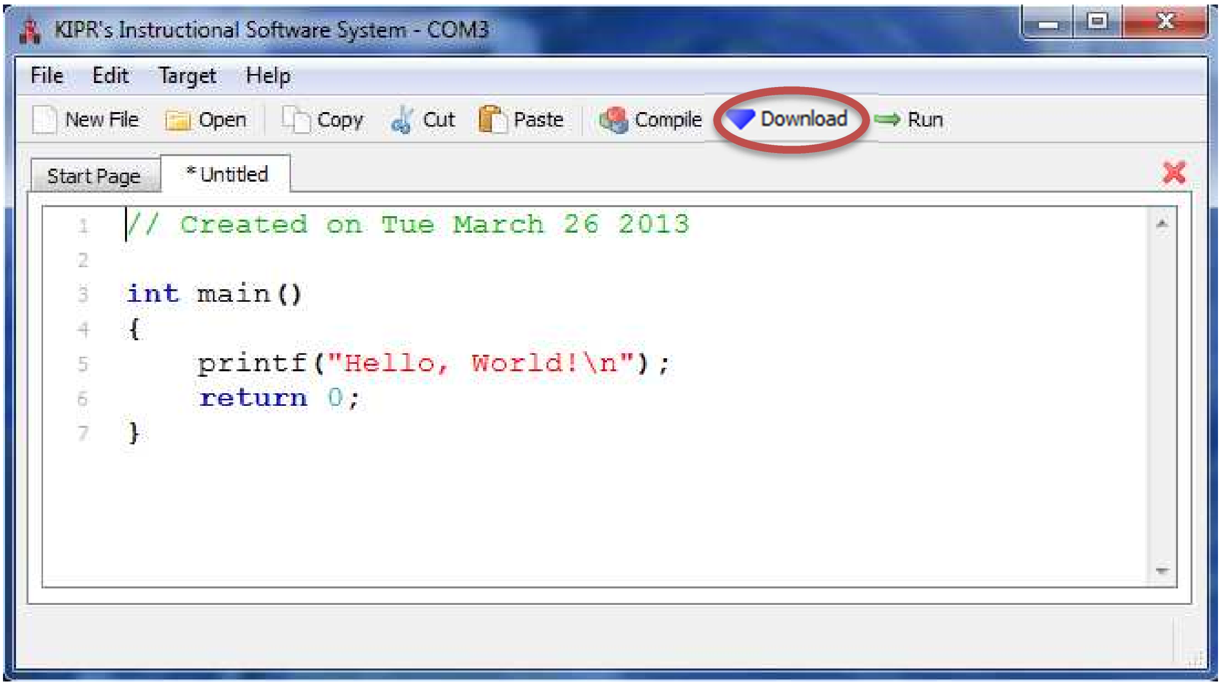
Show targets communicating over: All Interfaces

Next, the **Port Selection** dialog will appear. Under Os X, the KIPR Link will be detected as a USB modem and in Windows it will appear as pictured.

Choose COMx (where x is a number) from the Target Selection list and click OK. If no USB modem or COM port appears, make sure the CBC is powered on and click refresh once.

## 4. Download your program to the CBC

Next **click on the Download button** at the top left on the tool bar to transfer your program to the CBC.

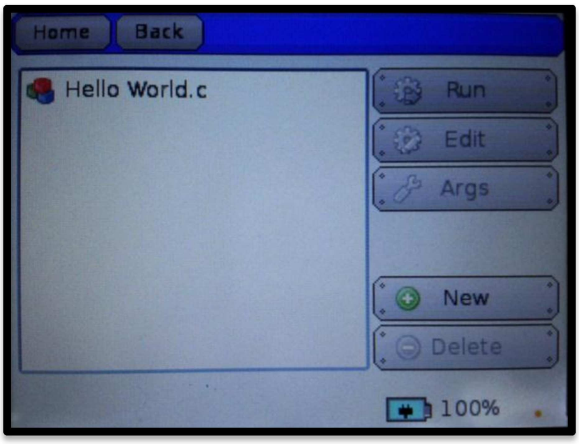


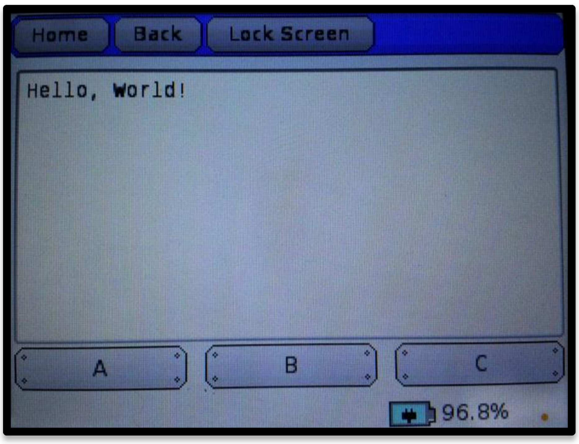
```
// Created on Tue March 26 2013

int main()
{
     printf("Hello, World!\n");
     return 0;
}
```

If you have not already done so, you will be prompted to save your file. Save it as "HelloWorld.c". After you save, the status line at the bottom left of the window will display Downloading, then Download Succeeded.

When the KIPR Link has finished compiling, press the Programs button on the main menu then select your program and press Run.





```
Hello, World!
```

The Program might compile, if it does press OK when it finishes. The red Stop button will be in the top right of the screen while the program is running and Hello World should be printed to the screen.

# 3. Programming the KIPR Link

## Using KISS-IDE as the KIPR Link IDE

KISS-IDE is the ANSI C programming environment used to program the KIPR Link. For more information on ANSI C programming pick up a ANSI C programming guide from your local book store. A highly recommended beginner's book is "Absolute Beginner's Guide to C" (2nd Edition) by Greg Perry.

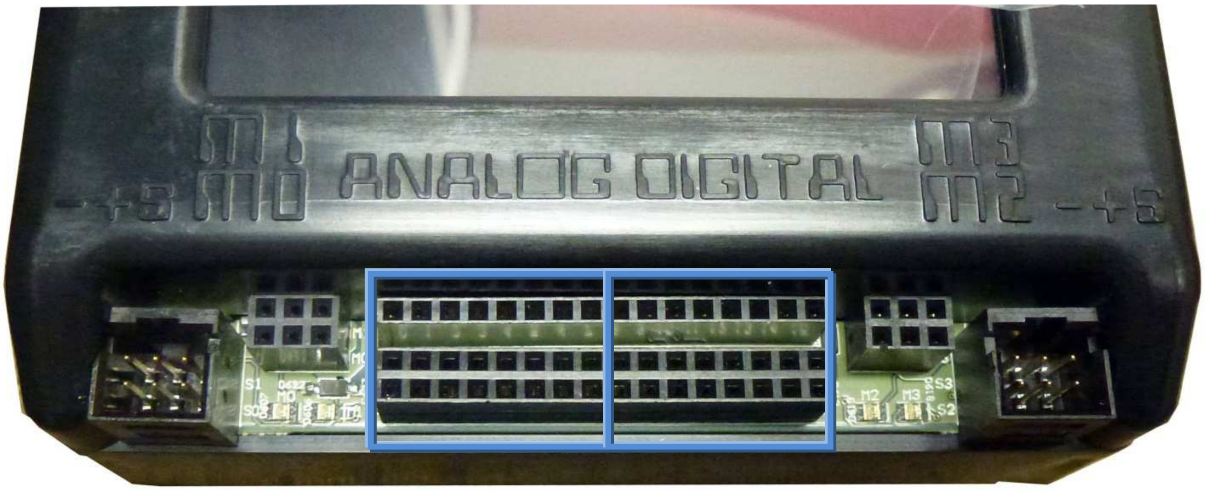
To write a program for the KIPR Link, plug in and turn on your KIPR Link. Launch KISS-IDE and follow steps 1 - 3 of the previous section "Download a Program to your KIPR Link".

Create a new file by clicking new or navigating to the File button and then New. A new program template will open. After choosing a template, you will need to pick a target. If you choose No Target you can write a program, but you will not be able to compile. Edit and add to the template to suit your needs. When you have a complete program you can hit the compile button to check for compiler errors. First your program will be saved (if you have not saved this program before you will be prompted to name the file and pick the save location), then any errors will be displayed at the bottom of the screen. If you double click on an error, you will be taken to the line of the error, but remember that some errors have to do with previous lines of code. To download a program to the KIPR Link, click the download button at the top. First your program will be saved, then compiled to check for errors, and finally downloaded to the KIPR Link. You will not be able to download a program that does not compile.

## KIPR Link Functions

KISS-IDE has functions for interacting with the KIPR Link. Some of the more common functions are presented in the following sections, and the complete list can be found in the Appendix.

# Using Sensors



Any sensors purchased from the Botball store will work with the KIPR Link. They are “keyed” so that there is only one orientation that all of the pins will be in holes.

Digital sensors typically only have two wires and are wired such that when the sensor is triggered the **SEN** and **GND** lines complete a circuit. Analog sensors can have two or three wires. In an analog sensor the resistance between the **SEN** and **GND** lines is varied. The third wire is connected to **VCC** which powers the sensor.

The default voltage on the KIPR Link from **VCC** to **GND** is 5V. See the appendix on how to change the **SEN** (sensor) voltage from 5V to 3.3V.

The specifications for creating your own sensor are in the appendix.

## KISS-C Sensor Library Functions

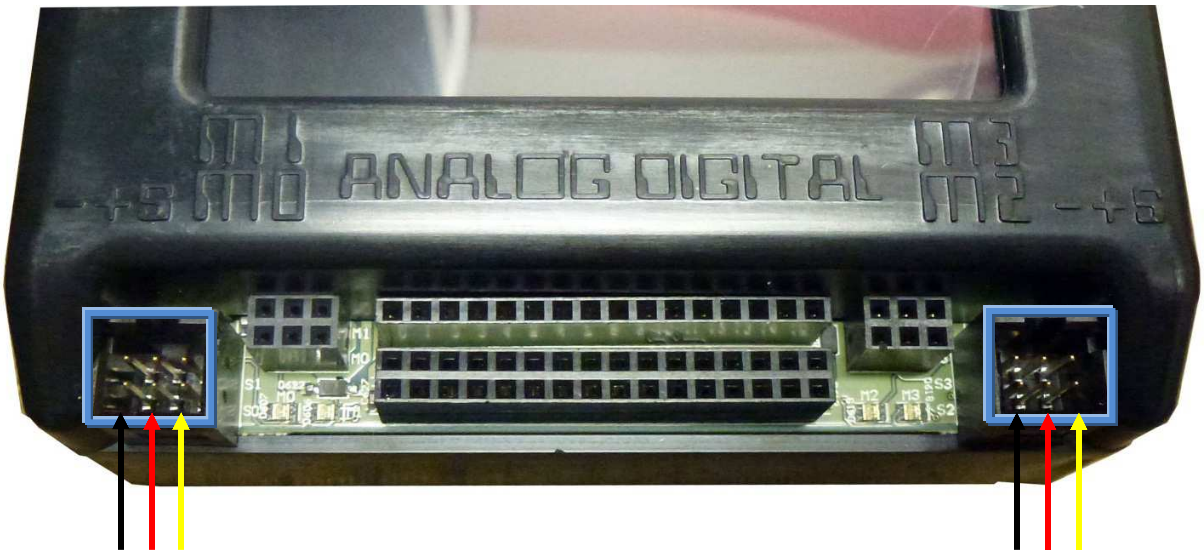
**analog10(<port#>)**

Returns the analog value of the port (a value in the range 0-1023). Analog ports are numbered 0-7. Light sensors and range sensors are examples of sensors you would use in analog ports.

**digital(<port#>)**

Returns 0 if the switch attached to the port is open and returns 1 if the switch is closed. Digital ports are numbered 8-15. Typically used for bumpers or limit switches.

## Using Servos



Servos plug in to the servo ports on the front of the KIPR Link. The arrows used above represent the most common coloring for servo cables (ground is black, positive is red, and signal is yellow), but yours may differ. If it does check the pin out to make sure that it is compatible with the KIPR Link before plugging it in. The servos run at 6 V. Plugging the servos in backwards will not cause damage, but the servos will not function properly until plugged in properly.

## KISS-C Servo Library Functions

**enable_servos()**

Enables power to the servo ports. This must be called before set_servo_position() to move a servo. When this function is called, the servo will default to position 1024 unless instructed to move elsewhere before enable_servos() is called.

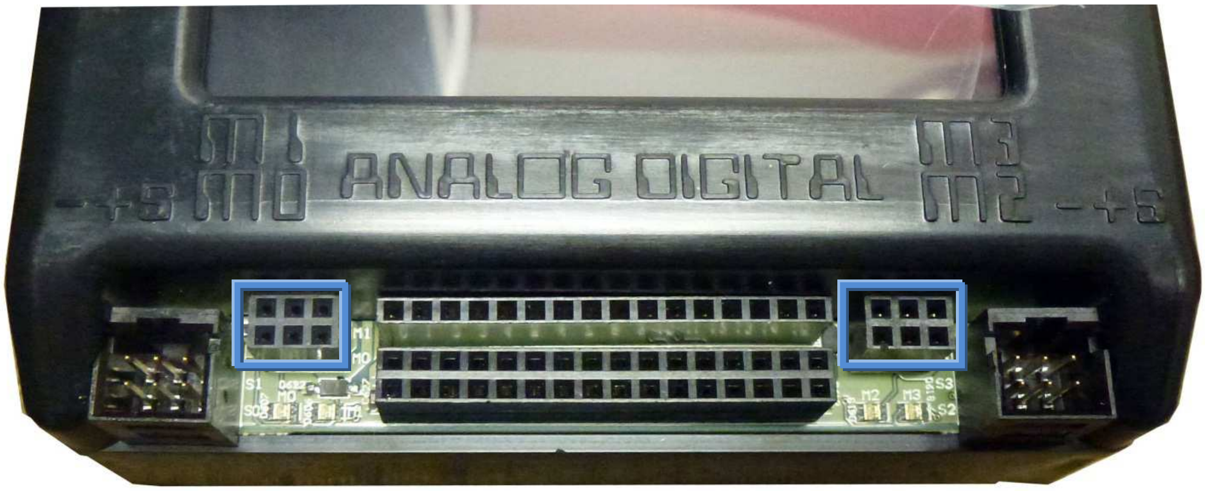
**disable_servos()**

Disables power to servo ports. This is useful when you want to conserve battery life. Your servo will be at the mercy of any external forces and will not hold its position when disabled.

**set_servo_position(<port#>, <position>)**

Moves a servo plugged in a port to a position. Position ranges from 0 to 2047. The servo will immediately move to position, unless impeded by external force. Call the enable_servos() function before using this function.

## Using Motors



The motors sold at the Botball Store will plug into any of the motor ports in any direction. When you instruct the motor to drive forward the motor will turn. If the motor is turning in the direction opposite of which you desire, un plug the motor, rotate the connector 180 degrees and plug the connector back in.

The motor ports run at 6V with a max current draw of 1A per motor port. Each group of ports (0 and 1, 2 and 3) is controlled by an h-bridge, so if you are going to be using close to 1A per motor, plug them into ports controlled by different h-bridges (i.e. 0 and 3) to prolong the life of your h-bridges.

Only the outside two positions of the connectors are used. When instructed to drive forward (green LED is lit), the position closest to the screen is ground and the position closest to the front is power. See the appendix for the specifications on creating your own motor plugs.

## KISS-C Motor Library Functions

**motor(<motor#>,<power>)**

Turns on a motor at a scaled PWM percentage. Power levels range from 100 (full forward) to -100 (full backward).

**mav(<motor#>,<velocity>)**

Move At Velocity moves a motor at a velocity indefinitely. Velocities range from -1000 to 1000 ticks per second.

**mrp(<motor#>,<velocity>,<position>)**

Move Relative Position moves a motor at a velocity from the current position to the current position plus the new position. Velocities range from 0 to 1000 ticks per second, and the position is a Long.

**bmd()**

Block Motor Done does not return until the previous mrp or mtp command finishes.

**ao()**

All Off turns off power to all motor ports.

## KISS-C Other Commonly Used Library Functions

**msleep(<seconds>)**

Returns after a time equal to or slightly greater than the argument in milliseconds.

**printf(<string>,<arg1>,<arg2>,…)**

Prints the string to the screen. If the string contains % codes then the arguments after the string will be printed in place of the % codes in the format specified by the code. See the Appendix for the full list of % codes.

# 5. Troubleshooting

If at any point you need additional help, are uncomfortable completing a troubleshooting step, or there is a problem you cannot resolve, call Technical Support at **(405) 579-4609** between 9AM and 5PM Central Standard Time, or email **support@kipr.org**.

| Problem | Solution |
|---|---|
| **My KIPR Link does not turn on.** | Plug it into the charger. A solid green LED should come on indicating the charger is correctly plugged in to power. A charge status LED should also come on indicating the amount of charge in the KIPR Link. If you get a flashing red LED, try un-plugging and re-plugging in the charger. If you continue to have the red blinking light, or a blinking green light, contact tech support. |
| **My KIPR Link comes on but only loads to the splash screen and keeps rebooting continuously.** | This behavior means that the Firmware is corrupted, or loaded incorrectly. Reload the most current version of the Firmware. See Appendix. |
| **My KIPR Link's touch screen is unresponsive, or it is difficult to press buttons in the user interface.** | You need to recalibrate the touch screen. Turn the KIPR Link off and back on. Allow the KIPR Link to boot to the main menu. From the main menu, select Settings and then Calibrate. If you cannot press those buttons, reload the firmware, as it will restore a default calibration. |
| **My KIPR Link is not recognized by my PC running Windows.** | Turn on your KIPR Link. Try reinstalling the driver from the KISS-IDE installer. Check to see if the KIPR Link is listed in the Device Manager. If there is a device warning, you are having issues installing the driver. If no device is present in the Device manager, check to make sure your USB cable is plugged in and functioning. If you are still having issues try a different USB port. |

# 6. Appendices

## Updating the Firmware

Follow the steps in the appropriate section below only if instructed to do so, or if the version numbers on your KIPR Link do not match the ones currently on:

http://www.kipr.org/products/link.

## Updating the Firmware

**Equipment Needed**: KIPR Link, KIPR Link AC Adapter, Flash Drive (a high quality one if possible)

Step 1: Download the most up to date version of the firmware file from:

http://www.kipr.org/products/link

Step 2: Unplug any motors, servos, or sensors from your KIPR Link and plug it in to charge from the AC adapter.

Step 3: Copy the kovan_update.img.gz file to the root (top level) of your USB flash drive.

Step 4: Un mount the USB drive from your computer. If the drive is not properly un mounted, the firmware file could get corrupted. In OSX right click on the drive and choose Eject. In Windows under My Computer right click on the drive and select Eject.

Step 5: Turn off the KIPR Link and plug the flash drive into a USB port.

Step 6: Press and hold the white button on the left side of the KIPR Link while turning on the power. Continue to hold the button until the screen displays the message Recovery Mode an Updating status bar.

Step 7: Once done the KIPR Link will reboot automatically. Then power it off and back on manually. The About screen should reflect the update, if not go to step 1.

## Enabling the pull up resistors on the analog ports in your program

By default all of the analog pull up resistors are disabled. Use the following built in function to enable or disable the pull up resistors.

**set_analog_pullup(int port, int state);**

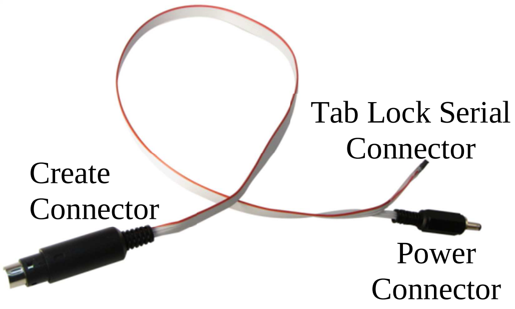
To enable a port set the state int to be 0; to disable a port, set the int to be 1. See the example below.

```
// disable pull up resistor on analog port 4 for ET
int main()
{
  set_analog_pullup(4,0);
  msleep(20);
}
```

In the example above, port 4 has the pull up resistors enabled.

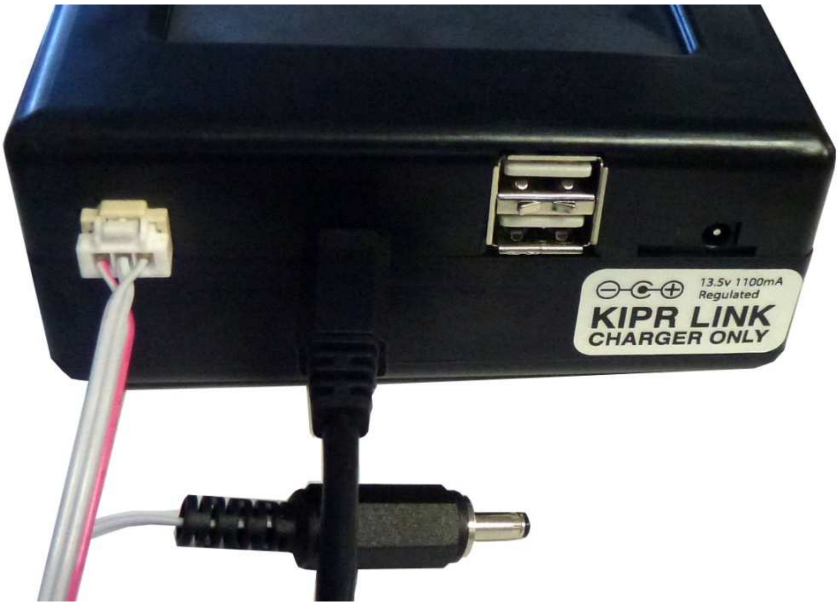
## Controlling an iRobot Create with the KIPR Link

The KIPR Link can communicate over a serial connection to control an iRobot Create. You will need to get a CBC Communication cable (available at https://botballstore.org). Once you have your CBC Communication cable follow the steps below to communicate with your iRobot Create.



Step 1: Connect the larger round plug of the CBC Communication cable to Mini-Din connection port in the iRobot Create (it is hidden under a removable cover above the charge port on the Create).

Step 2: Connect the locking header as shown below. The connector tab will lock in place, so you will need to depress the table before removing the cable. The power plug is optional, but when the iRobot Create is turned on your KIPR Link will trickle charge from the iRobot Create's battery.



Step 3: Write a program for the KIPR Link that communicates with the iRobot Create and download it to the KIPR Link.

Step 4: Set the iRobot Create on the ground.

Step 5: Power on the iRobot Create.

Step 6: Run your program on the KIPR Link.

## KISS-IDE iRobot Create Global Variables

These are the commonly called iRobot Create functions that report the sensor states, for a complete list, see the appendices.

**get_create_lbump()** – Left front bump sensor, 0 = no bump, 1 = bump

**get_create_rbump()** – Right front bump sensor, 0 = no bump, 1 = bump

**get_create_distance()** – Distance traveled since last requested in mm, between -32768 and 32768

**get_create_angle()** – Angle rotated since last request, clockwise is negative, between -32768 and 32768

**get_create_lcliff()** – Left cliff sensor, 0 = no cliff, 1 = cliff

**get_create_lfcliff()** – Left front cliff sensor, 0 = no cliff, 1 = cliff

**get_create_rfcliff()** – Right cliff sensor, 0 = no cliff, 1 = cliff

**gaet_create_rcliff()** – Right front cliff sensor, 0 = no cliff, 1 = cliff

## KISS-C iRobot Create Library Functions

These are the commonly used functions, for a complete list, see the appendices.

**create_connect();**

Connects the KIPR Link to the iRobot Create. Call this function first. By default the iRobot Create will be in "safe mode".

**create_disconnect();**

Disconnects the KIPR Link from the iRobot Create. Call this function at the end of the program, otherwise the KIPR Link will remain connected to the iRobot Create and if you rerun your program you will not be able to communicate with the iRobot Create.

**create_stop();**

Stops the drive wheels.

**create_drive(<speed>,<radius>);**

Drives in an arc at a set speed. Speed range is 20-500mm/s, radius is in mm.

**create_drive_direct(<left motor speed>,<right motor speed>);**

Specifies individual left and right motor speeds from 20-500 mm/s.

## Sample Program for Controlling an iRobot Create with the KIPR Link

Here is a program to make the iRobot Create drive in a circle with a radius of 0.25 meters at a speed of 200 mm/sec for 10 seconds and prints the distance traveled and the angle covered.

### Set Up

Fully charge your KIPR Link and iRobot Create. Connect the KIPR Link to the iRobot Create with the CBC Communication cable. Place the KIPR Link into the cargo bay of the iRobot Create. Set the iRobot Create on the floor with an adequate area cleaned out in front of the iRobot Create. Note that if the iRobot Create detects a ledge with the cliff or wheel drop sensors (like being picked up), the program will stop and the iRobot Create will play a sad tone.

### Code

```c
/* This is a program to make the iRobot Create drive in
a circle with a radius of 0.25 meters at a speed of 200
mm/sec for 10 seconds and prints the distance traveled
around the circle and the angle that the turn covered*/

int main()

{

create_connect();
set_create_distance(0);
set_create_normalized_angle(0);
create_drive(200, 250);
msleep(10000);
create_stop();
create_sensor_update();
printf("distance = %d\n", get_create_distance());
printf("angle = %d\n", get_create_normalized_angle());
create_disconnect();
return 0;

}
```

## KISS-C Library Functions for the KIPR Link

(alphabetic order)

a_button [Category: Sensors]
Format: int a_button();
Reads the value (0 or 1) of the A button.

alloff [Category: Motors]
Format: void alloff();
Turns off all motors. ao is a short form for alloff.

analog [Category: Sensors]
Format: int analog(int p);
Returns the value of the sensor installed at the port numbered p. The result is an integer between 0 and 255. The function can be used with analog ports 0 through 7.

analog10 [Category: Sensors]
Format: int analog10(int p);
10-bit version of the analog function. The returned value is in the range 0 to 1023 rather than 0 to 255.

ao [Category: Motors]
Format: void ao();
Turns off all motors.

atan [Category: Math]
Format: float atan(float angle);
Returns the arc tangent of the angle. Angle is specified in radians; the result is in radians.

b_button [Category: Sensors]
Format: int b_button();
Reads the value (0 or 1) of the B button.

bk [Category: Motors]
Format: void bk(int m);
Turns motor m on full speed in the backward direction.
Example:
bk(1);

black_button [Category: Sensors]
Format: int black_button();
Reads the value (0 or 1) of the Black button on the CBC (or a period on the simulator).

block_motor_done [Category: Motors]
Format: void block_motor_done(int m);
Function does not return until specified motor completes any executing speed or position control moves.
Example:
mrp(0,500,20000L);
block_motor_done(1);

bmd [Category: Motors]
Format: void bmd(int m);
Function does not return until specified motor completes any executing speed or position control moves.
Example:
mrp(0,500,20000L);
bmd(1);

display_clear [Category: Output]

Format: void display_clear();

Clear the KIPR Link display.

display_printf [Category: Output]

Format: void display_printf(int col, int row, char s[], . . .);

Perform a standard printf starting at screen location col, row.

c_button [Category: Sensors]

Format: int a_button();

Reads the value (0 or 1) of the A button.

clear_motor_position_counter [Category: Motors]

Format: void clear_motor_position_counter(int motor_nbr);

Reset the position counter for the motor specified to 0.

cos [Category: Math]

Format: float cos(float angle);

Returns cosine of angle. Angle is specified in radians; result is in radians.

defer [Category: Processes]

Format: void defer();

Makes a process swap out immediately after the function is called. Useful if a process knows that it will not need to do any work until the next time around the scheduler loop. defer() is implemented as a C built-in function.

digital [Category: Sensors]

Format: int digital(int p);

Returns the value of the sensor in sensor port p, as a true/false value (1 for true and 0 for false). Sensors are expected to be active low, meaning that they are valued at zero volts in the active, or true, state. Thus the library function returns the inverse of the actual reading from the digital hardware: if the reading is zero volts or logic zero, the digital() function will return true. Valid for digital ports 8-15.

disable_servos [Category: Servos]

Format: void disable_servos();

Disables the servo motor ports (powers down all servo motors).

enable_servos [Category: Servos]

Format: void enable_servos();

Enables all servo motor ports.

exp10 [Category: Math]

Format: float exp10(float num);

Returns 10 to the num power.

exp [Category: Math]

Format: float exp(float num);

Returns e to the num power.

fd [Category: Motors]

Format: void fd(int m);

Turns motor m on full in the forward direction.

Example:

fd(3);

freeze [Category: Motors]

Format: void freeze(int m);

Freezes motor m (prevents continued motor rotation, in contrast to off, which allows the motor to "coast").

get_motor_done [Category: Motors]

Format: int get_motor_done(int m);

Returns whether the motor has finished a move with specified position.

get_motor_position_counter [Category: Motors]

Format: int get_motor_position_counter(int m);

Returns the current motor position value for motor m (a value which is continually being updated for each motor using back EMF; a typical discrimination for a given motor is on the order of 1100 position "ticks" per rotation)

get_servo_position [Category: Servos]

Format: int get_servo_position(int srv);

Returns the position value of the servo in port srv. The value is in the range 0 to 2047. There are 4 servo ports (0, 1, 2, 3).

log10 [Category: Math]

Format: float log10(float num);

Returns the logarithm of num to the base 10.

log [Category: Math]

Format: float log(float num);

Returns the natural logarithm of num.

mav [Category: Motors]

Format: void mav(int m, int vel);

This function is the same as move_at_velocity

motor [Category: Motors]

Format: void motor(int m, int p);

Turns on motor m at scaled PWM duty cycle percentage p. Power levels range from 100 for full on forward to -100 for full on backward.

move_at_velocity [Category: Motors]

Format: void move_at_velocity(int m, int vel);

Moves motor m at velocity vel indefinitely. The velocity range is -1000 to 1000 ticks per second.

move_relative_position [Category: Motors]

Format: void move_relative_position(int m, int speed, int pos);

Moves motor m at velocity vel from its current position curr_pos to curr_pos + pos. The speed range is 0 to 1000 ticks per second.

Example:

```
move_relative_position(1,275,-1100L);
```

move_to_position [Category: Motors]

Format: void move_to_position(int m, int speed, int pos);

Moves motor m at velocity vel from its current position curr_pos to pos. The speed range is 0 to 1000. Note that if the motor is already at pos, the motor doesn't move.

mrp [Category: Motors]

Format: void mrp(int m, int vel, int pos);

This function is the same as move_relative_position.

mtp [Category: Motors]

Format: void mtp(int m, int vel, int pos);

This function is the same as move_to_position.

msleep [Category: Time]

Format: void msleep(int msec);

Waits for an amount of time equal to or greater than msec milliseconds.

Example:

```
/*wait for 1.5 seconds */ msleep(1500);
```

off [Category: Motors]

Format: void off(int m);
Turns off motor m.
Example:
off(1);

power_level [Category: Sensor]

Format: float power_level();
Returns the current power level in volts.

printf [Category: Output]

Format: void printf(char s[], . . .);
Prints the contents of the string referenced by s to the cursor position on the screen.

random [Category: Math]

Format: int random(int m);
Returns a random integer between 0 and some very large number.

run_for [Category: Processes]

Format: void run_for(float sec, void <function_name>);
This function takes a function and runs it for a certain amount of time in seconds. run_for will return within 1 second of your function exiting, if it exits before the specified time. The variable sec denotes how many seconds to run the given function.

set_analog_pullup [Category: Sensors]

Format: void set_analog_pullup (int port, int state);
This function uses a port number and the disabled state of the pullup 0 enabled, 1 for disabled.

set_pid_gains [Category: Motors]

Format: int set_pid_gains(int motor, int p, int i, int d, int pd, int id, int dd);
This function is used to adjust the weights of the PID control for the motors. The p, i and d parameters are the numerators for the p, i and d coefficients. The pd, id and dd parameters are their respective denominators. Thus all of the parameters are integers, but the actual coefficients can be floats. If a motor is jerky, the p and d terms should be reduced in size. If a motor lags far behind, they should be increased. The default values are 30,0,-30,70,1,51.

set_servo_position [Category: Servos]

Format: int set_servo_position(int srv, int pos);
Sets the position value of the servo in port srv. The value of pos must be in the range 0 to 2047. There are 4 servo ports (0, 1, 2, 3).

sin [Category: Math]

Format: float sin(float angle);
Returns the sine of angle. angle is specified in radians; result is in radians.

sqrt [Category: Math]

Format: float sqrt(float num);
Returns the square root of num.

tan [Category: Math]

Format: float tan(float angle);
Returns the tangent of angle. angle is specified in radians; result is in radians.

tone [Category: Output]

Format: void tone (float frequency, float length);
Produces a tone at pitch frequency (measured in Hertz) for length seconds. Returns when the tone is finished. Both frequency and length are floats.

## KISS-C Vision Library Functions for the KIPR Link

camera_open [Category: Vision API]

Format: void camera_open(<res>);
Opens camera for processing. The three resolutions are LOW_RES (160x120 – recommended), MED_RES (320x240), and HIGH_RES (640x480). The higher the resolution, the slower the tracking.

camera_update [Category: Vision API]

Format: void camera_update();
Processes tracking data for a new frame and makes it available for retrieval by the get_object_*property*() calls bellow.

get_object_area [Category: Vision API]

Format: int get_object_area(int ch, int i);
Returns the area of blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1) in pixels.

get_object_bbox .height [Category: Vision API]

Format: int get_object_bbox(int ch, int i).height;
Returns the area of blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1) in pixels.

get_object_bbox .width [Category: Vision API]

Format: int get_object_bbox(int ch, int i).width;
Returns the area of blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1) in pixels.

get_object_bbox .ulx [Category: Vision API]

Format: int get_object_bbox(int ch, int i).ulx;
Returns the pixel x coordinate of the upper leftmost pixel for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

get_object_bbox .uly [Category: Vision API]

Format: int get_object_bbox(int ch, int i).uly;
Returns the pixel y coordinate of the upper leftmost pixel for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

get_object_center .x [Category: Vision API]

Format: int get_object_center(int ch, int i).x;
Returns the pixel x coordinate of the centermost pixel for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

get_object_center .y [Category: Vision API]

Format: int get_object_center(int ch, int i).y;
Returns the pixel y coordinate of the centermost pixel for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

get_object_count [Category: Vision API]

Format: int get_object_count(int ch);
Returns the number of blobs from channel ch (range 0-3).

## KISS-C iRobot Create Library Functions for the KIPR Link

create_connect [Category: Create Function]

Format: int create_connect();
First step for connecting CBC to Create. Returns 0 if sucessful and a negative number if not. This function puts the Create in the create_safe mode.

create_disconnect [Category: Create Function]

Format: void create_disconnect();
Returns Create to proper state and resets XBC baud rate to KISS-C rate

create_start [Category: Create Function]

Format: void create_start();
Puts Create into active mode (with motors)

create_passive [Category: Create Function]

Format: void create_passive();
Puts Create into passive mode (no motors)

create_safe [Category: Create Function]

Format: void create_safe();
Create will execute all commands, but will disconnect and stop if drop or cliff sensors fire.

create_full [Category: Create Function]

Format: void create_full();
Create will move however you tell it -- even if that is a bad thing. In particular, the Create will not stop and disconnect, even if it is picked up or the cliff sensors fire.

create_spot [Category: Create Function]

Format: void create_spot();
Simulates a Roomba doing a spot clean

create_cover [Category: Create Function]

Format: void create_cover();
Simulates a Roomba covering a room

create_demo [Category: Create Function]

Format: void create_demo(int d);
Runs built in demos (see Create OI)

create_cover_dock    [Category: Create Function]

Format: void create_cover_dock();

Create roams around until it sees an IR dock and then attempts to dock

get_create_mode    [Category: Create Function]

Format: int get_create_mode();

Returns the Create's mode (0 off; 1 passive; 2 safe; 3 full).

get_create_lbump    [Category: Create Sensor Function]

Format: int get_create_lbump();

returns 1 if left bumper is pressed, 0 otherwise.

get_create_rbump    [Category: Create Sensor Function]

Format: int get_create_rbump();

returns 1 if right bumper is pressed, 0 otherwise.

get_create_lwdrop    [Category: Create Sensor Function]

Format: int get_create_lwdrop();

returns 1 if left wheel has dropped, 0 otherwise.

get_create_cwdrop    [Category: Create Sensor Function]

Format: int get_create_cwdrop();

returns 1 if caster wheel has dropped, 0 otherwise.

get_create_rwdrop    [Category: Create Sensor Function]

Format: int get_create_rlwdrop();

returns 1 if right wheel has dropped, 0 otherwise.

get_create_wall    [Category: Create Sensor Function]

Format: int get_create_wall();

returns 1 if wall is detected by right facing sensor, 0 otherwise.

get_create_lcliff    [Category: Create Sensor Function]

Format: int get_create_lcliff();

returns 1 if left cliff sensor is over black or a cliff, 0 otherwise.

get_create_lfcliff    [Category: Create Sensor Function]

Format: int get_create_lfcliff();

returns 1 if left front cliff sensor is over black or a cliff, 0 otherwise.

get_create_rfcliff [Category: Create Sensor Function]

Format: int get_create_rfcliff();
returns 1 if right frontcliff sensor is over black or a cliff, 0 otherwise.

get_create_rcliff [Category: Create Sensor Function]

Format: int get_create_rcliff();
returns 1 if right cliff sensor is over black or a cliff, 0 otherwise.

get_create_vwall [Category: Create Sensor Function]

Format: int get_create_vwall();
returns 1 if a virtual wall beacon is detected, 0 otherwise.

get_create_overcurrents [Category: Create Sensor Function]

Format: int get_create_overcurrents();
returns the overcurrent status byte where 16's bit indicates overcurrent in left wheel; 8's bit in right wheel, 4's bit is LD2, 2's bit is LD0 and 1's bit is LD1.

get_create_infrared [Category: Create Sensor Function]

Format: int get_create_infrared();
returns the byte detected from the remote control, 255 if no byte has been detected.

get_create_advance_button [Category: Create Sensor Function]

Format: int get_create_advance_button();
returns 1 if advance button is being pressed, 0 otherwise.

get_create_play_button [Category: Create Sensor Function]

Format: int get_create_play_button();
returns 1 if play button is being pressed, 0 otherwise.

get_create_distance [Category: Create Sensor Function]

Format: int get_create_distance();
returns the accumulated distance the Create has traveled since it was turned on or the distance was set. Moving backwards reduces this value. The distance is in millimeters. The value is truncated to the nearest millimeter every time this function is updated so having the lag time be too small will cause an artificially small value.

set_create_distance [Category: Create Sensor Function]
Format: void set_create_distance(int dist);
Sets the current value that will be returned by get_create_distance to the value dist.

get_create_normalized_angle [Category: Create Sensor Function]

Format: int get_create_normalized_angle();
returns the accumulated angle the Create has turned since it was turned on or the distance was set -- normalized to the range 0 to 359 degrees. Turning CCW increases this value and CW decreases the value. The value is truncated to the nearest degree every time this function is updated so having the lag time be too small will cause an artificially small value.

get_create_total_angle [Category: Create Sensor Function]

Format: int get_create_total_angle();
returns the accumulated angle the Create has turned since it was turned on or the distance was set. Turning CCW increases this value and CW decreases the value. The value is truncated to the nearest degree every time this function is updated so having the lag time be too small will cause an artificially small value.

set_create_normalized_angle [Category: Create Sensor Function]

Format: void set_create_normalized_angle(int angle);
Sets the current value that will be returned by get_create_normalized_angle to the value angle.

set_create_total_angle [Category: Create Sensor Function]

Format: void set_create_total_angle(int angle);
Sets the current value that will be returned by get_create_total_angle to the value angle.

get_create_battery_charging_state [Category: Create Sensor Function]

Format: int get_create_charging_state();
0-not charging; 1-recondition charging; 2-full charging; 3-trickle charging; 4-waiting; 5-charge fault.

get_create_battery_voltage [Category: Create Sensor Function]

Format: int get_create_battery_voltage();
returns the battery voltage in mV.

get_create_battery_current [Category: Create Sensor Function]

Format: int get_create_battery_current();
returns the current flow in mA.

get_create_battery_temp [Category: Create Sensor Function]

Format: int get_create_battery_temp();
returns the battery temperature in degrees C.

get_create_battery_charge [Category: Create Sensor Function]

Format: int get_create_battery_charge();
returns the battery charge in mAh.

get_create_battery_capacity [Category: Create Sensor Function]

Format: int get_create_battery_capacity();
returns the battery capacity in mAh.

get_create_wall_amt [Category: Create Sensor Function]

Format: int get_create_wall_amt();
returns 12 bit analog value from wall sensor.

get_create_lcliff_amt [Category: Create Sensor Function]

Format: int get_create_lcliff_amt();
returns 12 bit analog value from left cliff sensor.

get_create_lfcliff_amt [Category: Create Sensor Function]

Format: int get_create_lfcliff_amt();
returns 12 bit analog value from left front cliff sensor.

get_create_rfcliff_amt [Category: Create Sensor Function]

Format: int get_create_rfcliff_amt();
returns 12 bit analog value from right frontcliff sensor.

get_create_rcliff_amt [Category: Create Sensor Function]

Format: int get_create_rcliff_amt();
returns 12 bit analog value from right cliff sensor.

get_create_bay_DI [Category: Create Sensor Function]

Format: int get_create_bay_DI();
returns byte containing all digital sensors from the cargo bay: 16'bit for pin 15, 8's bit for pin 6, 4's bit for pin 18, 2's bit for pin 5 and 1's bit for pin 17.

get_create_bay_AI [Category: Create Sensor Function]

Format: int get_create_bay_AI();
returns 10 bit analog value on pin 4 from the cargo bay.

get_create_song_number [Category: Create Sensor Function]
Format: int get_create_song_number();
returns currently selected song 0-15.

get_create_song_playing [Category: Create Sensor Function]

Format: int get_create_song_playing();
returns 1 if song is playing, 0 otherwise.

get_create_number_of_stream_packets [Category: Create Sensor Function]

Format: int get_create_number_of_stream_packets();
if data streaming is being used, it returns the size of the stream.

get_create_requested_velocity [Category: Create Sensor Function]

Format: int get_create_requested_velocity();
asks Create how fast it was told to be moving -500 to 500mm/s and returns that value.

get_create_requested_radius [Category: Create Sensor Function]

Format: int get_create_requested_radius();
asks Create the size of its turning radius and returns that value.

get_create_requested_right_velocity [Category: Create Sensor Function]

Format: int get_create_requested_right_velocity();
asks Create how fast it was told to be moving right wheel and returns that value.

get_create_requested_left_velocity [Category: Create Sensor Function]

Format: int get_create_requested_left_velocity();
asks Create how fast it was told to be moving left wheel and returns that value.

create_stop [Category: Create Movement Function]

Format: void create_stop();
Stops the drive wheels

create_drive [Category: Create Movement Function]

Format: void create_drive(int speed, int radius);
Drives in an arc (see below for point turns and straight). Speed range for all commands is 20-500mm/s

create_drive_straight [Category: Create Movement Function]
Format: void create_drive_straight(int speed);
Drives straight at speed in mm/s

create_spin_CW [Category: Create Movement Function]

Format: void create_spin_CW(int speed);
spins Clockwise with edge speed of speed in mm/s

create_spin_CCW [Category: Create Movement Function]

Format: void create_spin_CCW(int speed);
spins Counterclockwise with edge speed of speed in mm/s

create_drive_direct [Category: Create Movement Function]

Format: void create_drive_direct(int r_speed, int l_speed);
Specifies individual left and right speeds in mm/s

create_advance_led [Category: Create Function]

Format: void create_advance_led(int on);
Pass 1 to turn on light and 0 to turn it off

create_play_led [Category: Create Function]

Format: void create_play_led(int on);
Pass 1 to turn on light and 0 to turn it off

create_power_led [Category: Create Function]

Format: void create_power_led(int color, int brightness);
color 0 is red and 255 green; brightness 0 is off and 255 is full brightness