Studuino Library

Function Reference

Published 2014/11/01
Revised 2018/02/01

Version History

Date	Content
2014/11/01	First version
2017/01/16	Updated for new Studuino website
2017/08/16	Changed Stduino.h to Studuino.h in program examples
2018/02/01	Revised text

Index

1. Getting Started 3
2. Functions 3
   - 2.1. Initialization Functions 3
   - 2.2. DC Motor Functions 7
   - 2.3. Servomotor Functions 10
   - 2.4. Buzzer Functions 12
   - 2.5. LED Functions 14
   - 2.6. Input Functions 15
   - 2.7. Timer Functions 22
   - 2.8. Studuino mini Functions 23
   - 2.9. Constants 28
3. Programming 31
   - 3.1. The Arduino Language 31
   - 3.2. Studuino Objects 31
   - 3.3. Including Header Files 31
   - 3.4. Sample Programs 32
     - 3.4.1. DC Motors 32
     - 3.4.2. Servomotors 33
     - 3.4.3. Buzzers 35
     - 3.4.4. LEDs 36
     - 3.4.5. Sensors 37

A. Connecting DC Motors to Studuino 42

1. Getting Started

This manual covers the Studuino Function Library used to describe programs made for your Studuino in Arduino IDE. The Library contains functions used to control ArtecRobo electronic parts including DC Motors and Servomotors, allowing you to use these functions to program your parts.

The information in this manual is subject to revision at any time.

2. Functions

Descriptions of functions will be in the following format:

Function name:	**(Function name)**			
Argument	(Type)	(Variable name)	(Value)	(Description)
Return value	(Type)	(Description)		
Notes				

Find descriptions of each function below.

2.1. Initialization Functions

The following section covers the functions used to initialize ports on your Studuino.

Function name	**SetDCMotorCalibration**			
Argument	byte[2]	rate	0 ~ 100	Offset value
Return value	None			

You'll only need to use this function to adjust your DC Motor's speed.

This function adjusts a DC Motor's speed. Change the values in argument **rate [0]** for M1 and **rate[1]** for M2 to control the corresponding DC Motor's speed.

(Example)

```
// Limit maximum speed of DC Motor on M1 to 80%
// Set maximum speed of DC Motor on M2 to 100%
byte calib[] = { 80, 100 };
SetDCMotorCalibration(calib); // DC Motor speed calibration setting
```

Function name	SetServomotorCalibration			
Argument	char[8]	offsets	-15 ~ 15	Offset values
Return values	None			

Use this function only to adjust a Servomotor's initial angle.

This function adjusts a Servomotor's angle. Change the values of the arguments in offsets **[0] ~ [7]** to specify the angle for the corresponding Servomotor on D2 ~ D12.

(Example)

```
// Specify a Servomotor connected to D9 ~ D12
byte calib[] = { 0, 0, 0, 0, -6, 0, 12, 3 };    //    Angles: D9(-6°), D10(0°), D11(12°), D12(3°)
SetServomotorCalibration(calib); // Servomotor speed calibration setting
```

Function name	InitDCMotorPort			
Argument	byte	connector	PORT_M1	Connector
			PORT_M2	
Return values	None			

This function initializes a DC Motor port. Use this function to initialize a port before using it with a DC Motor.

(Example)

```
// Use this function to initialize the port before running the Move, DCMotor, DCMotorPower, or DCMotorControl functions
InitDCMotorPort(PORT_M1); // Initialize port M1 for use with a DC Motor
```

Function name	InitServomotorPort			
Argument	byte	connector	See * 1	Connector
Return values	None			

This function initializes a Servomotor port. Use this function to initialize a port before using it with a Servomotor.

(Example)

```
// Use this function to initialize the port before running the Servomotor, SyncServomotors, or AsyncServomotors functions
InitServomotorPortPORT_D2); // Initialize port D2 for use with a Servomotor
```

Function name	InitServomotorPortForLED			
Argument	byte	connector	PORT_D9	Connector
			PORT_D10	
			PORT_D11	
Return values	None			

Initializes Servomotor ports D9, D10, or D11 for use with an LED. Use this function to initialize the port before adjusting the brightness of an LED on D9, D10, or D11.

(Example)

// Use this function to initialize the port before running the Gradation function

InitServomotorPortForLEDPORT_D9); // Initialize port D9 for use with an LED

Gradation(PORT_D9, 128);

Function name	InitSensorPort			
Argument	byte	connector	See * 4	Connector
	byte	pid	See * 5	Parts
Return values	None			

This function initializes a port for a Sensor, Buzzer, or LED. Use this function to initialize a port before using it with a Sensor, Buzzer, or LED.

(Example)

// Use this function to initialize the port before running the Buzzer, BuzzerControl, Melody, LED, or Get* functions

InitSensorPort(PORT_A0, PIDLED); // Initialize port A0 for use with an LED

Function name	**InitSensorPort**			
Argument	byte	connector1	See * 4	Connector
	byte	connector2	See * 4	Connector
	byte	pid	See * 5	Parts
Return values	None			
This function initializes ports for an Ultrasonic Sensor. Use this function to initialize ports before using them with an Ultrasonic Sensor. (Example) InitSensorPort(PORT_A0, PORT_A1, PIDULTRASONICSENSOR); // Initialize port A0 and A1 for use with an Ultrasonic Sensor				

Function name	**InitI2CPort**			
Argument	byte	pid	See * 5	Parts (I2C devices only)
Return values	None			
This function initializes the I2C port (A4, A5). Use this function to initialize the port before using it with an I2C device. (Example) InitI2CPort(PIDACCELEROMETER); // Initialize the I2C port for use with an Accelerometer				

Function name	**InitBluetooth**
Argument	None
Return values	None
Opens a serial connection via Bluetooth. The serial port is initialized at a baudrate of 9600. This function is supported by two Bluetooth Modules: RBT-001 Bluetooth Module (product #86873) Bluetooth Module for Robots (product #86876) (Example) InitBluetooth(); // Initialize Bluetooth	

2.2. DC Motor Functions

This section covers the functions used to control DC Motors.

<table>
<tr><td>Function name</td><td colspan="4">Move</td></tr>
<tr><td rowspan="12">Argument</td><td rowspan="8">byte</td><td rowspan="8">direct</td><td>FORWARD</td><td>Go forward</td></tr>
<tr><td>BACKWARD</td><td>Go backwards</td></tr>
<tr><td>FORWARD_RIGHT</td><td>Right turn (F)</td></tr>
<tr><td>FORWARD_LEFT</td><td>Left turn (F)</td></tr>
<tr><td>BACKWARD_RIGHT</td><td>Right turn (B)</td></tr>
<tr><td>BACKWARD_LEFT</td><td>Left turn (B)</td></tr>
<tr><td>CLOCKWISE</td><td>Turn clockwise</td></tr>
<tr><td>COUNTERCLOCKWISE</td><td>Turn counterclockwise</td></tr>
<tr><td>byte</td><td>pace</td><td>0 ~ 255</td><td>Speed (steps)</td></tr>
<tr><td>ulong</td><td>duration</td><td>0 ~ 2^32-1</td><td>Time (msec)</td></tr>
<tr><td rowspan="2">byte</td><td rowspan="2">brake</td><td>BRAKE</td><td>Use brakes</td></tr>
<tr><td>COAST</td><td>Don’t use brakes</td></tr>
<tr><td>Return value</td><td colspan="4">None</td></tr>
<tr><td colspan="5">This function is used to control the movements of a car that uses two DC Motors. The DC Motors will need to be attached to the board in a specific way in order to use it. See Connecting DC Motors to Studuino to learn how to build this car.
(Example)
Move (FORWARD, 10, 1000, BRAKE); // Car drives forward at speed 10 for one second before stopping</td></tr>
</table>

Function name	DCMotor			
Argument	byte	connector	PORT_M1	Connector
			PORT_M2	
	byte	rotation	NORMAL	Forward
			REVERSE	Backward
	byte	pace	0 ~ 255	Speed (steps)
	ulong	duration	0 ~ 2^32-1	Time (msec)
	byte	brake	BRAKE	Use brakes
			COAST	Don't use brakes
Return value	None			

This function controls a single DC Motor.

(Example)

// Make a DC Motor on M1 rotate at speed 10 for one second and stop

DCMotor (PORT_M1, NORMAL, 10, 1000, BRAKE);

Function name	DCMotorPower			
Argument	byte	connector	PORT_M1	Connector
			PORT_M2	
	byte	pace	0 ~ 255	Speed (steps)
Return value	None			

This function controls a DC Motor's speed.

(Example)

// Make a DC Motor on M1 rotate at speed 10 for one second, rotate at speed 100 for one second, and then stop

DCMotorPower(PORT_M1, 10); // Set the speed of the DC Motor on M1

DCMotorControl(PORT_M1, CLOCKWISE); // Set the DC Motor on M1 to clockwise

Timer(1000); // Count one second

DCMotorPower(PORT_M1, 100); // Change the speed of the DC Motor on M1

Timer(1000); // Count one second

DCMotorControl(PORT_M1, BRAKE); // Stop the DC Motor on M1

Function name	DCMotorControl			
Argument	byte	connector	PORT_M1	Connector
			PORT_M2	
	byte	rotation	NORMAL	Forward
			REVERSE	Backward
			BRAKE	Use brakes
			COAST	Don’t use brakes
Return value	None			

This function controls a DC Motor’s rotation.

(Example)

```
// Make a DC Motor on M1 rotate at speed 10 for one second and stop
DCMotorPower(PORT_M1, 10); // Set the speed of the DC Motor on M1
DCMotorControl(PORT_M1, CLOCKWISE); // Set the DC Motor on M1 to clockwise
Timer(1000); // Count one second
DCMotorControl(PORT_M1, BRAKE); // Stop the DC Motor on M1
```

2.3. Servomotor Functions

This section covers the functions used to control Servomotors.

Function name	Servomotor			
Argument	byte	connector	See * 1	Connector
	byte	degree	0 ~ 180	Servomotor Angle
Return value	None			
Sets an angle for one Servomotor. The next process in the program will run as the Servomotor is rotating. (Example) // Set the Servomotor on D2 to 90 degrees Servomotor (PORT_D2, 90);				

Function name	AsyncServomotors			
Argument	byte[]	connectors	See * 1	Connector arrangement
	byte[]	degrees	0 ~ 180	Angle for each Servomotor
	byte	number	1 ~ 8	Number of Servomotors
Return value	None			
Sets angles for multiple Servomotors. The next process in the program will run as the Servomotors are rotating. (Example) // Set the Servomotors on D2, D9, and D10 to 90, 180, and 45 degrees byte myConnectors[] = { PORT_D2, PORT_D9, PORT_D10 }; byte myDegrees[] = { 90, 180, 45}; ASyncServomotor (myConectors, myDegrees, 3);				

Function name	SyncServomotors			
Argument	byte[]	connector	See * 1	Arrangement of connectors
	byte[]	degree	0 ~ 180	Angle for each Servomotor
	byte	number	1 ~ 8	Number of Servomotors
	byte	time	3 ~ 255	Turning time per degree (ms)
Return value	None			

Sets angles for multiple Servomotors. The program will wait until all Servomotors rotate to the specified angles before running other processes. Make your Servomotors rotate more slowly by increasing the value of time. Be aware that the maximum speed per degree is three milliseconds, which means that setting time to any number under three will still make your Servomotors rotate at this speed.

(Example)

```
// Set the Servomotors on D2, D9, and D10 to 90, 180, and 45 degrees
byte myConectors[] = { PORT_D2, PORT_D9, PORT_D10 };
byte myDegrees[] = { 90, 180, 45};
SyncServomotor (myConectors, myDegrees, 3, 5);
```

2.4. Buzzer Functions

This section covers the functions used to control Buzzers.

Function name	**Buzzer**			
Argument	byte	connector	See * 3	Connector
	word	pitch	See * 6	Notes
	ulong	duration	0 ~ 2^32-1	Play time (msec)
Return value	None			
Plays the note from the Buzzer for the specified period of time. (Example) Buzzer (PORT_A0, BZR_C4, 1000); // Use Buzzer A0 to play the note “Do” for one second				

Function name	**BuzzerControl**			
Argument	byte	connector	See * 3	Connector
	boolean	onoff	ON	Play sound
			OFF	Stop sound
	byte	pitch	See * 6	Notes
Return value	None			

Setting the argument onoff to ON will make the Buzzer play a note at the specified pitch. Setting it to OFF will ignore the argument pitch and stop the Buzzer.
(Example)
// Use Buzzer A0 to play the note “Do” for one second
BuzzerControl(PORT_A0, ON, BZR_C4);
Timer(1000);
BuzzerControl(PORT_A0, OFF, 0);				

<table>
<tr><td>Function name</td><td colspan="4">Melody</td></tr>
<tr><td rowspan="7">Argument</td><td>byte</td><td>connector</td><td>See * 3</td><td></td></tr>
<tr><td>word[]</td><td>pitches</td><td>See * 6</td><td>Note</td></tr>
<tr><td>float[]</td><td>beats</td><td>0 ~</td><td>Beat</td></tr>
<tr><td>byte</td><td>number</td><td>Melody number</td><td>Number of notes</td></tr>
<tr><td rowspan="4">byte</td><td rowspan="4">tempo</td><td>TEMPO60</td><td rowspan="4">Tempo</td></tr>
<tr><td>TEMPO90</td></tr>
<tr><td>TEMPO120</td></tr>
<tr><td>TEMPO150</td></tr>
<tr><td>Return value</td><td colspan="4">None</td></tr>
<tr><td colspan="5">Uses the Buzzer to play a melody.
(Example)
// Use Buzzer A0 to play Do, Re, Mi, Fa, Mi, Re, Do.
word myPitches[] = { BZR_C3, BZR_D3, BZR_E3, BZR_F3,
BZR_E3, BZR_D3, BZR_C3 };
float myBeats[] = { 1, 1, 1, 1, 1, 1, 1 };
byte num = 7; // Number of elements
Melody (PORT_A0, myPitches, myBeats, num, TEMPO90);</td></tr>
</table>

2.5. LED Functions

This section covers the functions used to control LEDs.

Function name	**LED**			
Argument	byte	connector	See * 3	Connector
	boolean	onoff	ON	LED ON/OFF
			OFF	
Return value	None			
Turns the LED on or off. (Example) LED (PORT_A0, ON); // Turn an LED on A0 on				

Function name	**Gradation**			
Argument	byte	connector	PORT_D9	Connector
			PORT_D10	
			PORT_D11	
	byte	ratio	0 ~ 255	Brightness (higher values are brighter)
Return value	None			
Adjusts the brightness of an LED connected to port D9, D10, or D11. (Example) Gradation (PORT_D9, 128); // Set the brightness of an LED on D9				

2.6. Input Functions

This section covers functions used for Push-buttons and sensors.

Function name	**GetPushSwitchValue**			
Argument	byte	connector	See * 2	Connector
Return value	byte	0: Pressed, 1: Released		

Retrieves the state of a Push-button.
(Example)
// Get the value of a Push-button on A0
byte val = GetPushSwitchValue (PORT_A0);				

Function name	**GetTouchSensorValue**			
Argument	byte	connector	See * 3	Connector
Return value	byte	0: Pressed, 1: Released		

Retrieves the state of a Touch Sensor.
(Example)
// Get the value of a Touch Sensor on A0
byte val = GetTouchSensorValue (PORT_A0);				

Function name	**GetLightSensorValue**			
Argument	byte	connector	See * 4	Connector
Return value	int	0 ~ 1023		
Retrieves the value of a Light Sensor. (Example) int val = GetLightSensorValue (PORT_A0); // Get the value of a Light Sensor on A0				

Function name	**GetSoundSensorValue**			
Argument	byte	connector	See * 4	Connector
Return value	int	0 ~ 1023		
Retrieves the value of a Sound Sensor. (Example) int val = GetSoundSensorValue (PORT_A0); // Get the value of a Sound Sensor on A0				

Function name	GetIRPhotoreflectorValue			
Argument	byte	connector	See * 4	Connector
Return value	int	0 ~ 1023		

Retrieves the value of an IR Photoreflector.
(Example)
// Get the value of an IR Photoreflector on A0
int val = GetIRPhotoreflectorValue (PORT_A0);				

Function name	GetAccelerometerValue			
Argument	byte	axis	X_AXIS	Direction of measured acceleration
			Y_AXIS	
			Z_AXIS	
Return value	int	-128 ~ 127		
Retrieves the value of an Accelerometer. Accelerometers can only be used with the I2C port on A4/A5. (Example) int val = GetAccelerometerValue (X_AXIS); // Get Accelerometer tilt along X-axis				

Function name	GetTemperatureSensorValue			
Argument	byte	connector	See * 5	Connector
Return value	int	0 ~ 1023		

Retrieves the value of a Temperature Sensor.
(Example)
int val = GetTemperatureSensorValue(PORT_A0); // Get the value of a Temperature Sensor on A0
double temperature = ((((val / 1024.0) * 3.3) - 0.5) / 0.01); // Convert temperature to celsius				

Function name	GetUltrasonicSensorValue			
Argument	byte	trigger	See * 4	Connector
	byte	echo	See * 4	Connector
Return value	unsigned long		Time it takes to pickup ultrasonic waves (in microseconds)	

Retrieves the value of an Ultrasonic Sensor. It returns the amount of time (in microseconds) it takes for the ultrasonic waves emitted by the triggerPin to be picked up by the echoPin.
(Example)
unsigned long val = GetUltrasonicSensorValue(PORT_A0, PORT_A1); // Get the value of the Ultrasonic Sensor
double dist = val / 58.0; // Convert distance to cm
58 = 29[us/cm] * 2: Multiplies the amount of time (in microseconds) it takes for sound to travel one centimeter by two (making one round-trip)				

Function name	GetGyroscopeValue			
Argument	byte	axis	X_AXIS	Direction of measured acceleration
			Y_AXIS	
			Z_AXIS	
			GX_AXIS	Direction of measured angular velocity
			GY_AXIS	
			GZ_AXIS	
Return value	int	-32768 ~ 32767		
Retrieves acceleration and angular velocity of a Gyroscope. Gyroscopes can only be used with the I2C port on A4/A5. (Example) int val = GetGyroscopeValue(GX_AXIS); // Get the angular velocity along the Gyroscope's X-axis				

Function name	GetIRReceiverValue	
Argument	None	
Return value	unsigned long	IR Signal value
Retrieves the value of an IR Receiver. (Example) unsigned long val = GetIRReceiverValue(); // Get the value of the IR Receiver		

Function name	DisableIRReceiver
Argument	None
Return value	None

Disables the IR Receiver. You'll need to use this function to disable your IR Receiver before using a DC Motor on M1 or a Buzzer in the same program.
(Example)
board.DisableIRReceiver(); // Disable the IR Receiver
board.Move(BACKWARD, DCMPWR(10), 500, COAST); // Reverse
board.EnableIRReceiver(); // Enable the IR Receiver	

Function name	EnableIRReceiver
Argument	None
Return value	None

Enables the IR Receiver. You'll need to use this function to enable the IR Receiver after using DisableIRReceiver to disable it.
(Example)
board.DisableIRReceiver(); // Disable the IR Receiver
board.Move(BACKWARD, DCMPWR(10), 500, COAST); // Reverse
board.EnableIRReceiver(); // Enable the IR Receiver	

Function name	GetColorSensorValue			
Argument	byte	axis	VALUE_RED	Composition of measured color
			VALUE_GREEN	
			VALUE_BLUE	
			VALUE_CLEAR	
Return value	Unsigned int		Composition of detected color	
Retrieves the value of a Color Sensor. Returns the color composition of detected light by using red, green, blue, and clear filters (filterless). (Example) unsigned int sv = GetColorSensorValue(VALUE_RED); // Get the composition of red detected by the Color Sensor				

Function name	GetColorSensorXY		
Argument	double*	x	Used to store x color coordinates
	double*	y	Used to store y color coordinates
Return value	None		

Retrieves result of converting the color composition detected by the Color Sensor into color coordinates. These results are stored as x and y values.
(Example)
double x, y; // Declare variable to store values
GetColorSensorXY(&x, &y); // Get coordinates and store them to x and y			

Function name	GetColorCode		
Argument	None		
Return value	byte	COLOR_UNDEF	Undefinable color
		COLOR_RED	Red
		COLOR_GREEN	Green
		COLOR_BLUE	Blue
		COLOR_WHITE	White
		COLOR_YELLOW	Yellow
		COLOR_BROWN	Brown
		COLOR_BLACK	Black

Determines whether the Artec Block is red, green, blue, white, yellow, brown, or black and returns the corresponding code.

(Example)

```
int sv = board.GetColorCode(); // Return the color code of the Artec Block
```

Function name	UpdateBluetooth	
Argument	None	
Return value	boolean	Data receipt status

Attempts to receive data from the Bluetooth Controller application, returning TRUE if it receives data and FALSE if it doesn't.

(Example)

```
board.UpdateBluetooth();
boolean fID1 = board.GetBTCommandIDState(BT_ID_01); // Check whether the button assigned to ID01 in the application is pressed
if(fID1 == true) {
   // If button is pressed
}
```

Function name	GetBTCommandIDState		
Argument	byte	id	BT_ID_01 ~ BT_ID_10, BT_ID_ACC
Return value	boolean	Check whether ID and accelerometer are enabled	

Retrieves the state of the specified command ID from the value of UpdateBluetooth.

Target	ID	Return Value
On-screen Button	BT_ID_01 ~ BT_ID_10	TRUE: ON FALSE: OFF
Accelerometer	BT_ID_ACC	TRUE: Enabled FALSE: Disabled

(Example)

```
board.UpdateBluetooth();
boolean fID1 = board.GetBTCommandIDState(BT_ID_01); // Check whether the button assigned to ID01 in the application is pressed
if(fID1 == true) {
   // If button is pressed
}
```

Function name	GetBTAccelValue			
Argument	byte	axis	X_AXIS	Direction of measured acceleration
			Y_AXIS	
			Z_AXIS	
Return value	None			

Retrieves the value of the accelerometer after using UpdateBluetooth to receive data.

(Example)

```
board.UpdateBluetooth();
boolean fAcc = board.GetBTCommandIDState(BT_ID_ACC); // Determine whether the accelerometer is enabled inside of the application
int sv = board.GetBTAccelValue(X_AXIS); // Get device's X acceleration
if(fAcc & (sv > 0)) {
   // Process runs when accelerometer is enabled and X acceleration is true
}
```

2.7. Timer Functions

This section covers functions that wait for a specified time.

Function name	**Timer**			
Argument	unsigned long	time	0 ~ 2^32-1	Time (msec)
Return value	None			
Make the process stop for the specified amount of time. (Example) Timer(1000); // Stop for one second.				

2.8. Studuino mini Functions

The following functions can only be used with Studuino mini.

Function name	**InitClock**
Argument	None
Return value	None
Initializes the port for the LCD Clock.	

Function name	**setTime**			
Argument	byte	hour	0 ~ 23	Hour
	byte	min	0 ~ 59	Minute
Return value	None			
Sets the time of the LCD Clock. (Example) board.setTime(9, 0);				

Function name	**setDate**			
Argument	unsigned int	year	2000 ~ 2040	Year
	byte	month	1 ~ 12	Month
	byte	day	1 ~ 31	Day
Return value	None			
Sets the date of the LCD Clock. (Example) board.setDate(2016, 4, 1);				

Function name	**setAlarm**			
Argument	byte	hour	0 ~ 23	Hour
	byte	min	0 ~ 59	Minute
Return value	None			
Sets an alarm for the LCD Clock. (Example) board.setAlarm(9, 0);				

Function name	**setBackLight**			
Argument	byte	red	0 ~ 15	16 levels from 0 [dark] to 15 [bright]
	byte	green	0 ~ 15	16 levels from 0 [dark] to 15 [bright]
	byte	blue	0 ~ 15	16 levels from 0 [dark] to 15 [bright]
Return value	None			
Changes the color of the LCD Clock's backlight. (Example) board.setBackLight(15, 10, 5); // Set red to 15, green to 10, and blue to five				

Function name	**backLight**			
Argument	boolean	on/off	ON	Turns on backlight
			OFF	Turns off backlight
Return value	None			

Turns LCD Clock's backlight on or off.
(Example)
board.setBackLight(15, 10, 5); // Set red to 15, green to 10, and blue to five
board.backLight(ON); // Turn on backlight
board.Timer(1000); // Wait one second
clock.backLight(OFF); // Turn off backlight
 	
★ The clock will turn on using the color last specified in setBackLight.				

Function name	**clockBuzzer**			
Argument	word	pitch	See * 6	Notes
	unsigned long	duration		Output time (milliseconds)
Return value	None			
Plays a note using the LCD Clock's Buzzer. (Example) board.clockBuzzer(BZR_CS4, 2000);				

Function name	**GetHour**	
Argument	None	
Return value	int	Hour
Retrieves the hour from the LCD Clock.		

Function name	**GetMinute**	
Argument	None	
Return value	int	Minute
Retrieves the minutes from the LCD Clock.		

Function name	**GetYear**	
Argument	None	
Return value	int	Year
Retrieves the year from the LCD Clock.		

Function name	**GetMonth**	
Argument	None	
Return value	int	Month
Retrieves the month from the LCD Clock.		

Function name	**GetDay**	
Argument	None	
Return value	int	Day
Retrieves the day from the LCD Clock.		

Function name	**GetTemperature**	
Argument	None	
Return value	float	Temperature (℃)
Retrieves the temperature from the LCD Clock.		

Function name	**GetAlarmHour**	
Argument	None	
Return value	int	Alarm hour
Retrieves the alarm hour from the LCD Clock.		

Function name	**GetAlarmMinute**	
Argument	None	
Return value	int	Alarm minute
Retrieves alarm minute from the LCD Clock.		

Function name	**isAlarmTime**	
Argument	None	
Return value	boolean	Checks whether the current time matches the alarm time.

Returns whether the time of the LCD Clock matches the alarm time. The values returned by this function are linked to the LCD Clock's onboard alarm switch and STOP button.

	TRUE	FALSE
pitches	See * 3	・Always False
On	・Current time matches alarm time	・Current time doesn't match alarm time ・STOP button is pressed when True
Snooze	・Current time matches alarm time ・Current time is equal to alarm time + multiple of 5	・Current time doesn't match alarm time ・STOP button is pressed when True

(Example)

```
for(;;) {
  if(board.isAlarmTime()) {
    board.clockBuzzer(BZR_CS4, 2000);
  }
}
```

Function name	**sleep**	
Argument	None	
Return value	None	
SLEEP_MODE_PWR_SAVE will activate the clock’s power-saving mode.		

Function name	**GetOnboardLightSensor**	
Argument	None	
Return value	int	0 ~ 1023
Retrieves the value of the onboard Light Sensor.		

Function name	**GetBatteryVoltage**	
Argument	None	
Return value	float	Voltage [V]
Retrieves the voltage of the batteries connected to the board.		

2.9. Constants

*** 1 Servomotor Port Settings**

Value	Port
PORT_D2	D2
PORT_D4	D4
PORT_D7	D7
PORT_D8	D8
PORT_D9	D9
PORT_D10	D10
PORT_D11	D11
PORT_D12	D12

***2 Push-button Port Settings**

Value	Port
PORT_A0	A0
PORT_A1	A1
PORT_A2	A2
PORT_A3	A3

***3 Digital Port Values**

Value	Port
PORT_A0	A0
PORT_A1	A1
PORT_A2	A2
PORT_A3	A3
PORT_A4	A4
PORT_A5	A5

***4 Analog Port Values**

Value	Port
PORT_A0	A0
PORT_A1	A1
PORT_A2	A2
PORT_A3	A3
PORT_A4	A4
PORT_A5	A5
PORT_A6	A6
PORT_A7	A7

***5 Part IDs**

Value	Part
PIDOPEN	Disconnected
PIDLED	LED
PIDBUZZER	Buzzer
PIDLIGHTSENSOR	Light Sensor
PIDSOUNDSENSOR	Sound Sensor
PIDIRPHOTOREFLECTOR	IR Photoreflector
PIDACCELEROMETER	Accelerometer
PIDTOUCHSENSOR	Touch Sensor
PIDPUSHSWITCH	Push-button
PIDIRRECEIVER	IR Receiver
PIDGYROSCOPE	Gyroscope (*)
PIDTEMPERATURESENSOR	Temperature Sensor
PIDULTRASONICSENSOR	Ultrasonic Sensor
PIDCOLORSENSOR	Color Sensor (*)

(*): I2C device

*6 Note Values

Value	Note	Hz
BZR_C3	Do	130
BZR_CS3	Do #	139
BZR_D3	Re	147
BZR_DS3	Re #	156
BZR_E3	Mi	165
BZR_F3	Fa	175
BZR_FS3	Fa #	185
BZR_G3	So	196
BZR_GS3	So #	208
BZR_A3	La	220
BZR_AS3	La #	233
BZR_B3	Ti	247
BZR_C4	Do	262
BZR_CS4	Do #	277
BZR_D4	Re	294
BZR_DS4	Re #	311
BZR_E4	Mi	330
BZR_F4	Fa	349
BZR_FS4	Fa #	370
BZR_G4	So	392
BZR_GS4	So #	415
BZR_A4	La	440
BZR_AS4	La #	466
BZR_B4	Ti	494

Value	Note	Hz
BZR_C5	Do	523
BZR_CS5	Do #	554
BZR_D5	Re	587
BZR_DS5	Re #	622
BZR_E5	Mi	659
BZR_F5	Fa	698
BZR_FS5	Fa #	740
BZR_G5	So	784
BZR_GS5	So #	831
BZR_A5	La	880
BZR_AS5	La #	932
BZR_B5	Ti	988
BZR_C6	Do	1047
BZR_CS6	Do #	1109
BZR_D6	Re	1175
BZR_DS6	Re #	1245
BZR_E6	Mi	1319
BZR_F6	Fa	1397
BZR_FS6	Fa #	1480
BZR_G6	So	1568
BZR_GS6	So #	1661
BZR_A6	La	1760
BZR_AS6	La #	1865
BZR_B6	Ti	1976

Value	Note	Hz
BZR_C7	Do	2093
BZR_CS7	Do #	2217
BZR_D7	Re	2349
BZR_DS7	Re #	2489
BZR_E7	Mi	2637
BZR_F7	Fa	2794
BZR_FS7	Fa #	2960
BZR_G7	So	3136
BZR_GS7	So #	3322
BZR_A7	La	3520
BZR_AS7	La #	3729
BZR_B7	Ti	3951
BZR_C8	Do	4186
BZR_S	Silence	0

3. Programming

You'll have to keep the points in these sections in mind when using the Studuino Function Library to create a program for your Studuino.

3.1. The Arduino Language

The user has to define the setup and loop functions themselves when using the Arduino language. The setup function is only called once at the start of the program. Loop functions are used to repeat a defined process an infinite number of times.

```
// Called once at the start of the program. Primarily used for initialization.
void setup() {
   // Uses initialization functions to initialize Studuino ports with parts connected
}

// This function runs an infinite loop. Main process.
void loop() {
}
```

3.2. Studuino Objects

In order to use the Studuino library, you'll have to create an image of your Studuino board by placing Studuino Objects in a global variable.

```
// Studuino board image. Make only one per program.
Studuino board;       ★ For Studuino
StuduinoMini board;       ★ For Studuino mini
```

3.3. Including Header Files

These Library Functions are used by your Servomotors, Accelerometers, Gyroscopes, IR Receivers, and Color Sensors. You'll have to include these header files in addition to the header file for your Studuino.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>            // Servomotor header file
#include <Wire.h>             // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>        // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file
```

3.4. Sample Programs

The following sections contains example programs for each part.

3.4.1. DC Motors

Using A. Connecting DC Motors to Studuino as a reference, connect DC Motors to M1 and M2 on your Studuino and load the following program using the Arduino IDE.

The car will move forward for one second then reverse for one second.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>              // Servomotor header file
#include <Wire.h>               // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>         // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

// Called once at the start of the program. Primarily used for initialization.
void setup() {
   // Use initialization functions to initialize Studuino ports with parts connected
   board.InitDCMotorPort(PORT_M1);      // Initializes port M1 for use with a DC Motor.
   board.InitDCMotorPort(PORT_M2);      // Initializes port M2 for use with a DC Motor.
}

// This function runs an infinite loop. Main process.
void loop() {
   board.Move(FORWARD, 254, 1000, BRAKE);         // Drive forward for one second and stop
   board.Move(FORWARD, 254, 1000, BRAKE);         // Reverse for one second and stop

   // Rotate DC Motor M1 clockwise for one second and stop
   board.DCMotorPower(PORT_M1, 254);              // Set the speed of DC Motor M1
   board.DCMotorControl(PORT_M1, NORMAL);         // Begin rotating DC Motor M1 clockwise
   board.Timer(1000);                             // Wait one second
   board.DCMotorControl(PORT_M1, BRAKE);          // Stop DC Motor M1

   for (;;) {}       // Infinite loop to keep program from starting over from the top
}
```

3.4.2. Servomotors

Connect Servomotors to D10, D11, and D12 on your Studuino and load the following program using the Arduino IDE. All Servomotors will initialize at 90°. The program will wait three seconds before simultaneously turning all three Servomotors to 90°, 180°, and 0°. It will then wait three more seconds before turning Servomotor D10 to 180°.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>              // Servomotor header file
#include <Wire.h>               // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>        // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

// Called once at the start of the program. Primarily used for initialization.
void setup() {
   // Use initialization functions to initialize Studuino ports with parts connected
   board.InitDCMotorPort(PORT_D10);     // Initialize port D10 for use with a Servomotor
   board.InitDCMotorPort(PORT_D11);     // Initialize port D11 for use with a Servomotor
   board.InitDCMotorPort(PORT_D12);     // Initialize port D12 for use with a Servomotor
}
// This function runs an infinite loop. Main process.
void loop() {
   // Initialize Servomotor at 90 degrees
   byte connector[] = { PORT_D10, PORT_D11, PORT_D12 };
   byte degree[] = { 90, 90, 90 };
   byte number = sizeof(connector) / sizeof(byte); // Number of ports for which angles will
be set

   board.AsyncServomotors(connector, degree, number);
   // User may use this to add a delay until their Servomotors rotate
   board.Timer(1000);

   board.Timer(3000);          // Wait 3 seconds

   // Set Servomotors connected to ports D10, D11, and D12 to 90, 180, and 0 degrees
   degree[0] = 90;
   degree[1] = 180;
   degree[2] = 0;

   // Servomotors will have reached their target angle once this function finishes running

SyncServomotors(connector, degree, number, 10);

   board.Timer(3000);          // Wait 3 seconds

   // Set Servomotors connected to port D10 to 180 degrees
   board.Servomotor(PORT_D10, 180);
   // User may use this to add a delay before their Servomotors rotate
```

```
    board.Timer(1000);


    for (;;) {}        // Infinite loop to keep program from starting over from the top
}
```

3.4.3. Buzzers

Connect a Buzzer to A0 on your Studuino and load the following program using the Arduino IDE. This program plays “Do” for one second and “So” for one second before playing Twinkle, Twinkle, Little Star.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>             // Servomotor header file
#include <Wire.h>              // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>        // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

//These functions are run once at the start of the program. Primarily used for initialization.
void setup() {
   // Use initialization functions to initialize Studuino ports with parts connected
   board.InitDCMotorPort(PORT_A0 ,DC ); // Initialize port A0 for use with a Buzzer
}

// This function runs an infinite loop. Main process.
void loop() {
   // Play a one second note from the Buzzer
   board.Buzzer(PORT_A0, BZR_C5, 1000);

   board.Timer(1000);          // Wait 1 second

   // Play a one second note from the Buzzer
   board.BuzzerControl(PORT_A0, ON, BZR_G5);
   board.Timer(1000);
   board.BuzzerControl(PORT_A0, OFF, 0);// Ignore the last argument when set to OFF

   board.Timer(1000);          // Wait 1 second

   // Play a melody from the Buzzer
   word myPitches[] = { BZR_C5, BZR_C5, BZR_G5, BZR_G5, BZR_A5, BZR_A5, BZR_G5 };
   byte number = sizeof(myScales) / sizeof(word); // Number of notes to play
   float myBeats[] = { 1, 1, 1, 1, 1, 1, 1 };     // Tempo
   board.Melody (PORT_A0, myPitches, myBeats, number, TEMPO90);

   for (;;) {}        // Infinite loop to keep program from starting over from the top
}
```

3.4.4. LEDs

Connect LEDs to A1 and D9 on your Studuino and load the following program using the Arduino IDE. LED A1 will blink three times before LED D9 slowly turns on.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>             // Servomotor header file
#include <Wire.h>              // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>        // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

// Called once at the start of the program. Used primarily for initialization.
void setup() {
   // Use initialization functions to initialize Studuino ports with parts connected
   board.InitDCMotorPort(PORT_A1 ,DC );// Initialize port A0 for use with an LED
   board.InitDCMotorPort(PORT_D9);      // Initialize port D9 for use with an LED
}

// This function runs an infinite loop. Main process.
void loop() {
   // Blink LED A1 three times
   for (int i = 0;i < 3;i++) {
     board.LED(PORT_A1, ON);   // Turn on LED A1
     board.Timer(1000);        // Wait one second
     board.LED(PORT_A1, OFF);  // Turn off LED A1
     board.Timer(1000);        // Wait one second
   }

   // Slowly turn on LED D9
   board.Gradation(PORT_D9, 0);
   for (int i = 0;i < 255;i++) {
     board.Gradation(PORT_D9, i);
     board.Timer(100);
   }

   for (;;) {}        // Infinite loop to keep program from starting over from the top
}
```

3.4.5. Sensors

① Regular Sensors

Connect a Touch Sensor to A1, a Sound Sensor to A2, an IR Photoreflector to A3, an Accelerometer to A4/A5, and a Light Sensor to A6 on your Studuino and load the following program using the Arduino IDE. Once you've loaded the program, go to Tools in Arduino IDE and choose Serial Monitor to open the Serial Monitor. The Serial Monitor shows you the value of each sensor.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>             // Servomotor header file
#include <Wire.h>              // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>        // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

// Called once at the start of the program. Used primarily for initialization.
void setup() {
   // Use initialization functions to initialize Studuino ports with parts connected
   board.InitSensorPort(PORT_A0, PIDPUSHSWITCH);
   board.InitSensorPort(PORT_A1, PIDTOUCHSENSOR);
   board.InitSensorPort(PORT_A2, PIDSOUNDSENSOR);
   board.InitSensorPort(PORT_A3, PIDIRPHOTOREFLECTOR);
   board.InitSensorPort(PORT_A4, PIDACCELEROMETER);
   board.InitSensorPort(PORT_A5, PIDACCELEROMETER);
   board.InitSensorPort(PORT_A6, PIDLIGHTSENSOR);

   // Begin serial output
   Serial.begin(9600);
}

// This function runs an infinite loop. Main process.
void loop() {
   // Retrieve Light Sensor value every 100 msec and output to Serial Monitor
   for (;;) {
     byte pVal = board.GetPushSwitchValue(PORT_A0);
     byte tVal = board.GetTouchSensorValue(PORT_A1);
     int  sVal = board.GetSoundSensorValue(PORT_A2);
     int  iVal = board.GetIRPhotoreflectorValue(PORT_A3);
     int  xVal = board.GetAccelerometerValue(X_AXIS);
     int  yVal = board.GetAccelerometerValue(Y_AXIS);
     int  zVal = board.GetAccelerometerValue(Z_AXIS);
     int  lVal = board.GetLightSensorValue(PORT_A6);
     Serial.print("button:");  Serial.print(pVal);        Serial.print("¥t");
     Serial.print("touch:");   Serial.print(tVal);        Serial.print("¥t");
     Serial.print("sound:");   Serial.print(sVal);        Serial.print("¥t");
     Serial.print("ir:");      Serial.print(iVal);        Serial.print("¥t");
     Serial.print("x:");       Serial.print(xVal);        Serial.print("¥t");
```

```
    Serial.print("y:");       Serial.print(yVal);        Serial.print("¥t");
    Serial.print("z:");       Serial.print(zVal);        Serial.print("¥t");
    Serial.print("light:");   Serial.print(lVal);        Serial.println();
    board.Timer(100);
  }
}
```

② Optional Parts: Ultrasonic Sensors, Temperature Sensors, and Gyroscopes

Connect an Ultrasonic Sensor to A0/A1, a Temperature Sensor to A2, and a Gyroscope to A4/A5 on your Studuino and load the following program using the Arduino IDE. Once you've loaded the program, go to Tools in Arduino IDE and choose Serial Monitor to open the Serial Monitor. The Serial Monitor shows you the value of each sensor.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>             // Servomotor header file
#include <Wire.h>              // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>        // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

// Called once at the start of the program. Used primarily for initialization.
void setup() {
  // Use initialization functions to initialize Studuino ports with parts connected
  board.InitSensorPort(PORT_A0, PORT_A1, PIDULTRASONICSENSOR);
  board.InitSensorPort(PORT_A2, PIDTEMPERATURESENSOR);
  board.InitI2CPort(PIDGYROSCOPE);

  // Begin serial output
  Serial.begin(9600);
}

// This function runs an infinite loop. Main process.
void loop() {
  // Retrieve Light Sensor value every 100 msec and output to Serial Monitor
  int uVal = board.GetUltrasonicSensorValue(PORT_A0, PORT_A1);
  int tVal = board.GetTemperatureSensorValue(PORT_A2);
  int xVal = board.GetGyroscopeValue(X_AXIS);
  int yVal = board.GetGyroscopeValue(Y_AXIS);
  int zVal = board.GetGyroscopeValue(Z_AXIS);
  int gxVal = board.GetGyroscopeValue(GX_AXIS);
  int gyVal = board.GetGyroscopeValue(GY_AXIS);
  int gzVal = board.GetGyroscopeValue(GZ_AXIS);
  Serial.print("ultrasonic:");         Serial.print(uVal);        Serial.print("¥t");
  Serial.print("temperature:");        Serial.print(tVal);        Serial.print("¥t");
  Serial.print("x:");         Serial.print(xVal);        Serial.print("¥t");
  Serial.print("y:");         Serial.print(yVal);        Serial.print("¥t");
  Serial.print("z:");         Serial.print(zVal);        Serial.print("¥t");
  Serial.print("gx:");        Serial.print(gxVal);       Serial.print("¥t");
```

```
    Serial.print("gy:");        Serial.print(gyVal);        Serial.print("¥t");
    Serial.print("gz:");        Serial.print(gzVal);        Serial.println();;
    board.Timer(100);
}
```

③ Optional Parts: IR Receivers

Connect an IR Receiver to A0 on your Studuino and load the following program using the Arduino IDE. Once you've loaded the program, go to Tools in Arduino IDE and choose Serial Monitor to open the Serial Monitor. The Serial Monitor shows you the value of each sensor.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>              // Servomotor header file
#include <Wire.h>               // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>        // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

// Called once at the start of the program. Used primarily for initialization.
void setup() {
   // Use initialization functions to initialize Studuino ports with parts connected
   board.InitSensorPort(PORT_A0, PIDIRRECEIVER);  // Sensor input setting

   // Begin serial output
   Serial.begin(9600);
}

// This function runs an infinite loop. Main process.
void loop() {
   // Retrieve IR Receiver value every 100 msec and output to Serial Monitor
   unsigned long ir = board.GetIRReceiverValue();
   if(ir != 0) {
     Serial.print("IR receive:");       Serial.print(ir, HEX);      Serial.println();
   }
   board.Timer(100);
}
```

④ Optional Parts: Color Sensors

Connect a Color Sensor to A4/A5 on your Studuino and load the following program using the Arduino IDE. Once you've loaded the program, go to Tools in Arduino IDE and choose Serial Monitor to open the Serial Monitor. The Serial Monitor shows you the value of each sensor.

```
#include <Arduino.h >// Basic header file
#include <Servo.h>             // Servomotor header file
#include <Wire.h>              // I2C device header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>        // Gyroscope header file
#include <IRremoteForStuduino.h>       // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h" // Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

// Called once at the start of the program. Used primarily for initialization.
void setup() {
   // Use initialization functions to initialize Studuino ports with parts connected
   board.InitI2CPort(PIDCOLORSENSOR);  // Start Color Sensor

   // Begin serial output
   Serial.begin(9600);
}

// This function runs an infinite loop. Main process.
void loop() {
   // Retrieve Color Sensor value every 100 msec and output to Serial Monitor
   unsigned int rVal = board.GetColorSensorValue(VALUE_RED);
   unsigned int gVal = board.GetColorSensorValue(VALUE_GREEN);
   unsigned int bVal = board.GetColorSensorValue(VALUE_BLUE);
   unsigned int cVal = board.GetColorSensorValue(VALUE_CLEAR);
   double x, y;
   board.GetColorSensorXY(&x, &y);

   Serial.print("red:");       Serial.print(rVal);         Serial.print("¥t");
   Serial.print("green:");     Serial.print(gVal);         Serial.print("¥t");
   Serial.print("blue:");      Serial.print(bVal);         Serial.print("¥t");
   Serial.print("clear:");     Serial.print(cVal);         Serial.print("¥t");
   Serial.print("X:");         Serial.print(x);   Serial.print("¥t");
   Serial.print("Y:");         Serial.print(y);   Serial.println();
   board.Timer(100);
}
```

⑤ Optional Parts: Bluetooth Module

Without connecting anything, load the following program using the Arduino IDE. Once it's loaded, plug the Bluetooth communication pins on D0 and D1 into the power supply of any open sensor port from A0-A7. Open the application, connect your Studuino, and send values to turn your Studuino's onboard LED on pin 13 on and off.

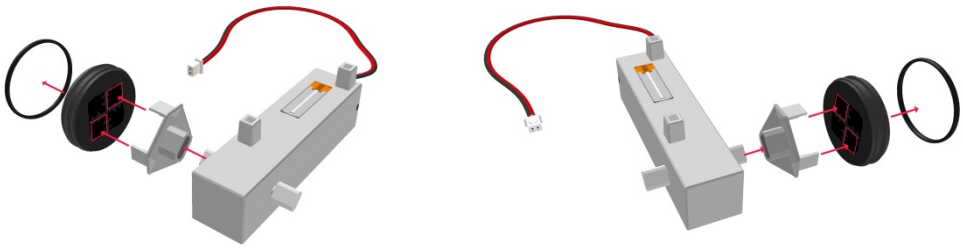
```
#include <Arduino.h >// Basic header file
#include <Servo.h>              // Servomotor header file
#include <Wire.h>               // Accelerometer header file
#include <MMA8653.h> // Accelerometer header file
#include <MPU6050.h>         // Gyroscope header file
#include <IRremoteForStuduino.h>        // IR Receiver header file
#include <ColorSensor.h>    // Color Sensor header file
#include "Studuino.h"// Studuino header file

// Studuino board image. Make only one per program.
Studuino board;

// Called once at the start of the program. Used primarily for initialization.
void setup() {
   // Use initialization functions to initialize Studuino ports with parts connected
   board.InitBluetooth();
   pinMode(13, OUTPUT);
}

// This function runs an infinite loop. Main process.
void loop() {
   // Get values from application
   board.UpdateBluetooth();

   // Turn LED on or off depending on received value
   if(board.GetBTCommandIDState(BT_ID_01)) {
     digitalWrite(13, HIGH);
   }
   if(board.GetBTCommandIDState(BT_ID_02)) {
     digitalWrite(13, LOW);
   }
   board.Timer(100);
}
```

A. Connecting DC Motors to Studuino

Follow the instructions below to assemble your car:

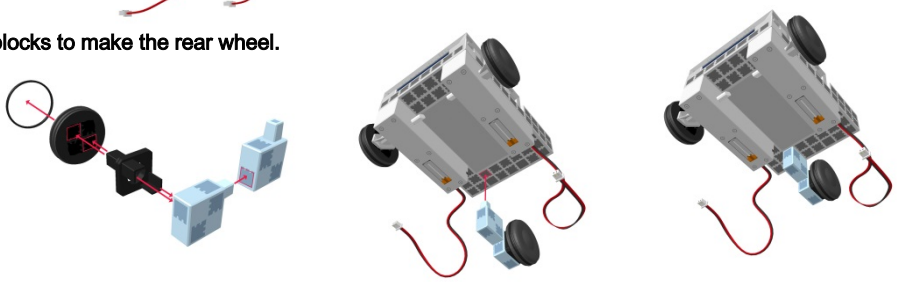
(1) Attach wheels to the DC Motors as shown below.

★ Make a symmetrical pair.

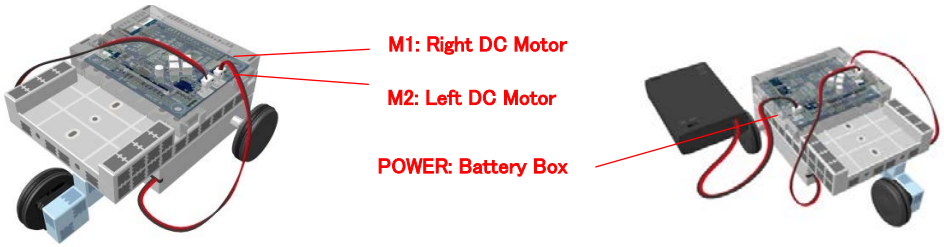


(2) Attach both DC Motors to the bottom of the Studuino mount.

(3) Use blocks to make the rear wheel.



(4) Now plug your DC Motors and Battery Box into your Studuino.



(5) Set your Battery Box in your Studuino mount to keep it in place.