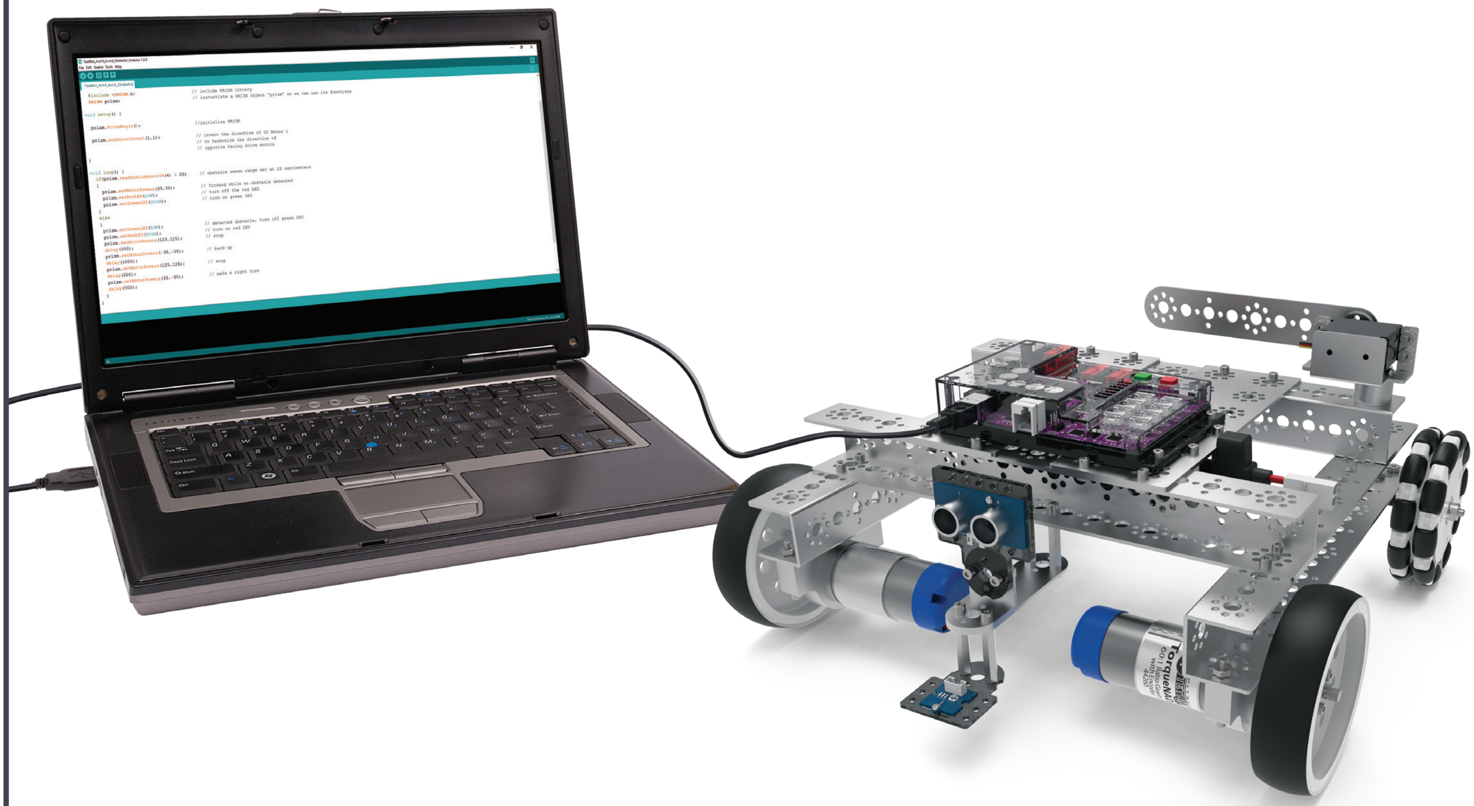

TETRIX® PRIZM® and *Arduino IDE* Reference Guide

Understanding common PRIZM commands and functions in the *Arduino Software (IDE)*

69955

Content creation by Aaron Locke.

Desktop publishing by Todd McGeorge.

©2018 Pitsco, Inc., 915 E. Jefferson, Pittsburg, KS 66762

All rights reserved. This product and related documentation are protected by copyright and are distributed under licenses restricting their use, copying, and distribution. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Pitsco, Inc.

All other product names mentioned herein might be the trademarks of their respective owners.

V1.0
06/18

Table of Contents

PRIZM Controller Commands 2
LED Commands 3
PRIZM Status Commands 3
DC Motor Commands 4
DC Motor Encoder Commands 5-7
Servo Commands 8-9
Sensor Commands 10
Logic Commands 11-12
Loop Commands 13-14
Variable Commands 15-16
Function Commands 17-18
Time Commands 19
Serial Monitor Commands 20

Command	Syntax	Purpose	Location in Sketch
Include the PRIZM library	#include <PRIZM.h>	• Includes the PRIZM library in the sketch so that Arduino can recognize PRIZM functions.	• Usually goes before the **void setup()** command.
Instantiate PRIZM	PRIZM prizm;	• Instantiates the PRIZM object so its functions can be used.	• Usually goes before the **void setup()** command.
PRIZM begin	prizm.PrizmBegin();	• Initializes the PRIZM controller.	• Usually is one of the first commands in the **void setup()** section. • Must come before any **prizm.** commands.
PRIZM end	prizm.PrizmEnd();	• Terminates the program running on PRIZM and resets the PRIZM controller.	• Usually is found somewhere in the main loop to stop the program. • At times it might be necessary to have this command in the **void setup()** section. ∘ For example, in the **void setup()** section, you could read the battery voltage, and if it is too low, you could stop the program before it has a chance to do anything else.

```
  #include <PRIZM.h>                      // include the PRIZM library
  PRIZM prizm;                            // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {
  prizm.PrizmBegin();                     // initialize the PRIZM controller
  if(prizm.readBatteryVoltage()<1000){    // if the battery voltage is less than 10.00 volts...
    prizm.PrizmEnd();                     // ...terminate the program
  }
}

void loop() {                             // loop forever
  prizm.setRedLED(HIGH);                  // turn the red LED on
  delay(1000);                            // wait here for 1000 ms (1 second)
  prizm.setRedLED(LOW);                   // turn the red LED off
  delay(1000);                            // wait here for 1000 ms (1 second)
}
```

This sketch shows the proper use and placement of the PRIZM controller commands.

Command	Syntax	Purpose	Location in Sketch
Turn on red LED	prizm.setRedLED(HIGH);	• Turn on PRIZM's red LED	• Usually, LED control commands are found in the **void loop()** section of a sketch. • In some situations, they could be used in the **void setup()** section before the sketch enters the main loop.
Turn off red LED	prizm.setRedLED(LOW);	• Turn off PRIZM's red LED	
Turn on green LED	prizm.setGreenLED(HIGH);	• Turn on PRIZM's green LED	
Turn off green LED	prizm.setGreenLED(LOW);	• Turn off PRIZM's green LED	

```
  #include <PRIZM.h>                    // include the PRIZM library
  PRIZM prizm;                          // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {
  prizm.PrizmBegin();                   // initialize the PRIZM controller
}

void loop() {                           // loop forever
  prizm.setRedLED(HIGH);                // turn the red LED on
  prizm.setGreenLED(LOW);               // turn the green LED off
  delay(1000);                          // wait here for 1000 ms (1 second)
  prizm.setRedLED(LOW);                 // turn the red LED off
  prizm.setGreenLED(HIGH);              // turn the green LED on
  delay(1000);                          // wait here for 1000 ms (1 second)
}
```

This sketch shows how the LED commands can be used to manipulate PRIZM's red and green LEDs.

Command	Syntax	Purpose	Location in Sketch
Read the state of PRIZM's Start button	prizm.readStartButton();	• Determines if PRIZM's green Start button is pressed or not. • Returns an integer value of 1 if pressed and 0 if not pressed.	• Can be used anywhere in a sketch.
Read the voltage of the battery attached to PRIZM	prizm.readBatteryVoltage();	• Determines the voltage of the battery attached to PRIZM. • Returns an integer value (ex: 915 = 9.15 volts).	• Can be used anywhere in a sketch.

```
int batteryVoltage = 1200;                        // set the variable batteryVoltage to 1200 for 12.00 volts
if(prizm.readStartButton() == 1){                 // if the Start button is pressed...
  batteryVoltage = prizm.readBatteryVoltage();    // ...set the variable batteryVoltage to the actual battery voltage
}
if(batteryVoltage < 900){                         // if the battery voltage is less than 9.00 volts...
  prizm.PrizmEnd();                               // ...terminate the program
}
```

Although this is not a full sketch, these example statements show how the battery voltage and Start button commands can be used.

Command	Syntax	Purpose	Location in Sketch
Set DC motor power	prizm.setMotorPower(*motor#*, *power*); *motor#* = 1 or 2 *power* = -100 to 100 or 125	• Sets the power and direction of a specified motor to make it rotate at a certain rate. • Range is from -100 to 100. • Value of 0 indicates a coasting stop. • Value of 125 indicates a hard-braking stop. • Negative values reverse the direction.	• Usually found in the **void loop()** section of a sketch. • Often found in called functions.
Set all DC motor powers simultaneously	prizm.setMotorPowers(*power1*, *power2*); *power* = -100 to 100 or 125	• Sets the power and direction of both motors to make them rotate at a certain rate. • Range is from -100 to 100. • Value of 0 indicates a coasting stop. • Value of 125 indicates a hard-braking stop. • Negative values reverse the direction.	• Usually found in the **void loop()** section of a sketch. • Often found in called functions.
Invert motor direction	prizm.setMotorInvert(*motor#*, 1); *motor#* = 1 or 2	• Inverts the rotation of the specified motor. • Is used when two motors oppose each other so that positive power values cause the same direction of motion. • Value of 1 indicates invert while 0 indicates no invert.	• Usually found in the **void setup()** section of a sketch.

```
  #include <PRIZM.h>                // include PRIZM library
  PRIZM prizm;                      // instantiate a PRIZM object "prizm" so we can use its functions
void setup() {
  prizm.PrizmBegin();               // initialize PRIZM
  prizm.setMotorInvert(1,1);        // invert the direction of DC Motor 1
}
void loop() {
  prizm.setMotorPowers(50,50);      // turn Motors 1 and 2 on at 50% power to drive forward
  delay(3000);                      // wait here for 3 seconds while motors are spinning
  prizm.setMotorPowers(0,0);        // stop both motors with a coasting stop
  delay (1000);                     // wait 1 second for robot to stop
  prizm.setMotorPower(1,-50);       // reverse one motor to pivot the robot
  delay(2000);                      // wait here for 2 seconds while the robot pivots
  prizm.setMotorPowers(30,30);      // turn Motors 1 and 2 on at 30% power to drive forward
  delay(3000);                      // wait here for 3 seconds while motors are spinning
  prizm.setMotorPowers(125,125);    // stop both motors with a hard stop
  prizm.PrizmEnd();                 // end program and reset PRIZM
}
```

This sketch uses DC motor commands to move a robot forward, pivot a robot, and stop a robot using different motor power levels.

Command	Syntax	Purpose	Location in Sketch
Set the rotational speed of a DC motor	prizm.setMotorSpeed(*motor#*, *speed*); *motor#* = 1 or 2 *speed* = -720 to 720 TorqueNADO accuracy: -630 to 630	• Precisely sets the speed of a motor in degrees per second. • Max speed is 720 degrees per second, or 2 rps. • For TorqueNADO®, a more accurate range is -630 to 630 degrees per second. • Rotational speed will be constant no matter the battery voltage.	• Usually found in the **void loop()** section of a sketch. • Often found in called functions.
Set the rotational speeds of both DC motors simultaneously	prizm.setMotorSpeeds(*speed1*, *speed2*); *speed* = -720 to 720 TorqueNADO accuracy: -630 to 630	• Precisely sets the speeds of both motors in degrees per second. • Max speed is 720 degrees per second, or 2 rps. • For TorqueNADO, a more accurate range is -630 to 630 degrees per second. • Rotational speed will be constant no matter the battery voltage.	• Usually found in the **void loop()** section of a sketch. • Often found in called functions.
Set a DC motor to a target position at a designated speed	prizm.setMotorTarget(*motor#*, *speed*, *target*); *motor#* = 1 or 2 *speed* = -720 to 720 *target* = -2147483648 to 2147483647	• Tells a motor to rotate a designated count at a designated speed and then stop in a holding position. • Each count represents 1/4 of a degree. • Maximum rotation is 2,147,483,647 counts, which is 536,870,912 degrees or 1,491,308 rotations.	• Usually found in the **void loop()** section of a sketch. • Often found in called functions.
Set both DC motors to target positions simultaneously at designated speeds	prizm.setMotorTargets(*speed1*, *target1*, *speed2*, *target2*); *speed* = -720 to 720 *target* = -2147483648 to 2147483647	• Tells both motors to rotate designated counts at designated speeds and then stop in a holding position. • Each count represents 1/4 of a degree. • Maximum rotation is 2,147,483,647 counts, which is 536,870,912 degrees or 1,491,308 rotations.	• Usually found in the **void loop()** section of a sketch. • Often found in called functions.

Command	Syntax	Purpose	Location in Sketch
Rotate a DC motor a designated number of degrees at a designated speed	prizm.setMotorDegree(*motor#*, *speed*, *degrees*); *motor#* = 1 or 2 *speed* = -720 to 720 *degrees* = -536870912 to 536870911	• Tells a motor to rotate a designated number of degrees at a designated speed and then stop in a holding position. • Maximum rotation is 536,870,911 degrees, which is 1,491,308 rotations.	• Usually found in the **void loop()** section of a sketch. • Often found in called functions.
Rotate both DC motors a designated number of degrees at designated speeds	prizm.setMotorDegrees(*speed1*, *degrees1*, *speed2*, *degrees2*); *speed* = -720 to 720 *degrees* = -536870912 to 536870911	• Tells both motors to rotate designated numbers of degrees at designated speeds and then stop in a holding position. • Maximum rotation is 536,870,911 degrees, which is 1,491,308 rotations.	• Usually found in the **void loop()** section of a sketch. • Often found in called functions.
Read motor busy status	prizm.readMotorBusy(*motor#*); *motor#* = 1 or 2	• Determines if a motor is busy carrying out a motor positioning task from another command. • Can be used to keep a program from moving to the next command until a motor is finished carrying out the previous positioning task. • Will return 1 for busy or 0 for hold position. • Eliminates the need to use a **delay()** command to wait for the motor to complete its task.	• Usually found in the **void loop()** section of a sketch or a called function. • Often follows a motor positioning command as part of a loop to wait for the motor to stop spinning.
Read encoder count	prizm.readEncoderCount(*encoder#*); *encoder#* = 1 or 2	• Reads the count value of the designated encoder. • Values range from -2,147,483,648 to 2,147,483,647. • Clockwise rotation adds to the count and counterclockwise rotation subtracts from the count. • Four encoder counts equal one degree of rotation. • 1,140 encoder counts equal one full motor rotation. • Encoder counts start at 0 at power-up and reset.	• Often found in conditional statements and loops that perform a task based on the encoder count.

Command	Syntax	Purpose	Location in Sketch
Read encoder degrees	prizm.readEncoderDegrees(*encoder#*); *encoder#* = 1 or 2	• Reads the degrees of rotation for the designated encoder. • Values range from -536,870,912 to 536,870,911. • Clockwise rotation adds to the count and counterclockwise rotation subtracts from the count. • 360 degrees equal one full motor rotation. • Encoder counts start at 0 at power-up and reset.	• Often found in conditional statements and loops that perform a task based on the encoder count.
Reset an encoder count	prizm.resetEncoder(*encoder#*); *encoder#* = 1 or 2	• Resets the encoder count back to 0.	• Often found after an encoder positioning task has completed and the counter needs to be reset for a new task.
Reset both encoder counts	prizm.resetEncoders();	• Resets the count for both encoders back to 0.	• Often found after encoder positioning tasks have completed and both encoders need to be reset for a new task.

```
#include <PRIZM.h>    // include the PRIZM library in the sketch
PRIZM prizm;          // instantiate a PRIZM object “prizm” so we can use its functions

int wheelDiam = 4;                             // diameter of a 4" standard TETRIX wheel
float wheelCirc = wheelDiam*M_PI;              // circumference of the wheel (c = pi * d) is dist traveled per motor rotation
float distPerDeg = wheelCirc/360;              // distance traveled per degree of rotation
float distPerEncCount = wheelCirc/1440;        // distance traveled per encoder count
int degPerFoot = round(12/distPerDeg);         // degrees of rotation to travel 1 foot
int countPerFoot = round(12/distPerEncCount);  // number of encoder counts to travel 1 foot

void setup() {
  prizm.PrizmBegin();                          // initialize the PRIZM controller
  prizm.setMotorInvert(1,1);                   // invert Motor 1
}

void loop() {
  prizm.setMotorDegrees(360,degPerFoot,360,degPerFoot);        // using degrees, move forward 1 foot at 1 rps
  while(prizm.readMotorBusy(1)==1){}                           // do nothing until the robot is finished moving
  delay (1000);                                                // wait 1 second
  prizm.setMotorDegrees(180,0,180,0);                          // using degrees, move backward 1 foot at 0.5 rps
  while(prizm.readMotorBusy(1)==1){}                           // do nothing until the robot is back to where it started
  delay (3000);                                                // wait 3 seconds
  prizm.setMotorTargets(360,-countPerFoot,360,-countPerFoot);  // using encoder count, move backward 1 foot at 1 rps
  while(prizm.readMotorBusy(1)==1){}                           // do nothing until the robot is finished moving
  delay (1000);                                                // wait 1 second
  prizm.setMotorTargets(180,0,180,0);                          // using encoder count, move backward 1 foot at 0.5 rps
  while(prizm.readMotorBusy(1)==1){}                           // do nothing until the robot is back to where it started
  if(prizm.readEncoderCount(1) != 0){                          // because the robot is back where it started, the encoder...
    prizm.resetEncoders();                                     // ...count should be 0, but just in case it isn't, reset...
  }                                                            // ...both encoders
  prizm.setMotorSpeeds(90,-90);                                // pivot the robot
  delay(3000);
  prizm.setMotorSpeeds(0,0);                                   // stop the robot
  prizm.resetEncoders();                                       // reset the encoders and repeat the loop
}
```

This sketch uses encoders to accurately move a robot specific distances at specific speeds.

Command	Syntax	Purpose	Location in Sketch
Set servo motor speed	prizm.setServoSpeed(*servo#*, *speed*); *servo#* = 1 to 6 *speed* = 0 to 100	• Sets the speed of a single servo. • The servo will default to a max speed of 100% if the servo speed is not set. • Servo speed will remain the same throughout the program until changed.	• Often found in the **void setup()** section of a sketch to control the speed of a servo throughout the program.
Set all servo motor speeds simultaneously	prizm.setServoSpeeds(*speed1*, *speed2*, *speed3*, *speed4*, *speed5*, *speed6*); *speed* = 0 to 100	• Sets the speed of up to six possible servos attached to PRIZM at the same time. • Servos will default to a max speed of 100% if a servo speed is not set. • Servo speed will remain the same throughout the program until changed.	• Often found in the **void setup()** section of a sketch to control the speed of a servo throughout the program.
Set standard servo motor position	prizm.setServoPosition(*servo#*, *degrees*); *servo#* = 1 to 6 *degrees* = 0 to 180	• Sets the angular position of a single servo.	• Can be found in the **void setup()** section to set the initial position of a servo. • Most often used in the **void loop()** section of a sketch.
Set all standard servo motor positions simultaneously	prizm.setServoPositions(*degrees1*, *degrees2*, *degrees3*, *degrees4*, *degrees5*, *degrees6*); *degrees* = 0 to 180	• Sets the angular position of all standard servos attached to PRIZM at the same time.	• Can be found in the **void setup()** section to set the initial position of all servos. • Can be used in the **void loop()** section of a sketch to move multiple servos simultaneously.
Read standard servo position	prizm.readServoPosition(*servo#*); *servo#* = 1 to 6 returned value = integer 0 to 180	• Reads the last position that was sent to a servo and returns an integer value between 0 and 180. • This command doesn't read the actual position of the servo. It returns what was last commanded and assumes the servo made it to that position. • The actual servo position and returned value could be different depending on the servo speed and whether appropriate time was allowed for the servo to reach that position. • Useful for synchronizing two servos. • Useful for storing a servo position for use later.	• Most often used in the **void loop()** section of a sketch.

Command	Syntax	Purpose	Location in Sketch
Set continuous rotation (CR) state	prizm.setCRServoState(*CRservo#*, *state*); *CRservo#* = 1 or 2 *state* = -100, 0, or 100	• Sets the on/off condition and direction of a continuous rotation servo. • A state value of -100 is a counterclockwise spin. • A state value of 100 is a clockwise spin. • A state value of 0 is off.	• Most often used in the **void loop()** section of a sketch.

```
#include <PRIZM.h>    // include the PRIZM library in the sketch
PRIZM prizm;          // instantiate a PRIZM object “prizm” so we can use its functions

void setup() {
  prizm.PrizmBegin();                                       // initialize the PRIZM controller
  prizm.setServoSpeeds(50,50,50,50,50,50);                  // set all servo speeds to 50%
  prizm.setServoPositions(0,0,0,0,0,0);                     // set all servo positions to 0 degrees
}
void loop() {
  prizm.setServoPosition(1,180);                            // rotate Servo 1 to 180 degrees
  prizm.setServoPosition(2,(prizm.readServoPosition(1)/2)); // rotate Servo 2 to half of Servo 1's position
  prizm.setCRServoState(1,100);                             // rotate CR Servo 1 clockwise
  delay(3000);                                              // wait for 3 seconds for servos to move into position

  prizm.setServoPositions(0,0,0,0,0,0);                     // set all servo positions to 0 degrees
  prizm.setCRServoState(1,-100);                            // rotate CR Servo 1 counterclockwise
  delay(3000);                                              // wait for 3 seconds for servos to move into position
}
```

This sketch uses servo commands to move servos to different positions and to rotate a continuous rotation servo.

Command	Syntax	Purpose	Location in Sketch
Read line sensor	prizm.readLineSensor(*port#*) *port#* = D2-D5 or A1-A3	• Reads the digital output of the Line Finder Sensor. • Value of 0 means light was reflected (light-colored surface). • Value of 1 means light was not reflected (dark-colored surface). • Can be connected to any digital port (D2-D5) or any analog ports configured as digital inputs (A1-A3).	• Most often used in the **void loop()** section of a sketch. • Often used in a conditional or while loop.
Read ultrasonic sensor in centimeters	prizm.readSonicSensorCM(*port#*) *port#* = D2-D5 or A1-A3	• Is used to determine distance to an object. • Reads the output of the Ultrasonic Sensor in centimeters (3-400 cm) or inches (2-150 in.) as an integer. • Can be connected to any digital port (D2-D5) or any analog ports configured as digital inputs (A1-A3).	
Read ultrasonic sensor in inches	prizm.readSonicSensorIN(*port#*) *port#* = D2-D5 or A1-A3		

```
  #include <PRIZM.h>                    // include PRIZM library
  PRIZM prizm;                          // instantiate a PRIZM object “prizm” so we can use its functions
void setup() {
  prizm.PrizmBegin();                       // initialize PRIZM
  prizm.setMotorInvert(1,1);                // invert the direction of DC Motor 1
  prizm.setServoSpeed(1,50);                // set Servo 1 speed to 50
}
void loop() {
  if(prizm.readLineSensor(3) == 1){         // line detected
    prizm.setMotorPowers(30,125);           // turn right
    prizm.setRedLED(HIGH);                  // turn on the red LED
   }
   else {                                   // no line detected
    prizm.setMotorPowers(125,30);           // turn left
    prizm.setRedLED(LOW);                   // turn off the red LED
   }

  while(prizm.readSonicSensorCM(4) < 25){    // object is in path, loop here until object is cleared
    prizm.setGreenLED(HIGH);                 // turn on green LED
    prizm.setMotorPowers(125,125);           // stop the motors
    prizm.setServoPosition(1,0);             // raise the detection flag
  }
  prizm.setGreenLED(LOW);                    // turn off green LED
  prizm.setServoPosition(1,90);              // lower the detection flag
}
```

This sketch uses the Line Finder and Ultrasonic Sensors to follow a line until an obstacle is detected. When an obstacle is detected, the robot stops and waits for the obstacle to be removed before it continues following the line.

Command	Syntax	Purpose	Location in Sketch

If statement	if(*condition*){*do this;*}
	if(*condition*){
	do this;
	}
		• Conditional logic statement that tests for a single condition.
		• Runs a command or series of commands if a certain condition is true.
		• If the condition within the parentheses is true, then the statements within the brackets are run.
			• Most often used in the **void loop()** section of a sketch or in a called function.

If-else statement	if(*condition*){
	do this;
	}
	else {
	do this;
	}
		• Conditional logic statement that tests for a single condition.
		• Runs a command or series of commands if a certain condition is true and runs a different command or series of commands if the condition is false.	

If-else if statement	if(*condition1*){
	do this;
	}
	else if(*condition2*) {
	do this;
	}
	else {
	do this;
	}
		• Conditional logic statement that tests for multiple conditions.
		• Runs a command or series of commands depending on which condition is met.	

Comparison operators	equal to: == not equal to: != less than: < less than or equal to: <= greater than: > greater than or equal to: >=	• Is used in comparison expressions to compare one value/variable on the left with another value/variable on the right. • **Note:** A single equal sign is used to assign a value. Two equal signs are required for comparison.	• Usually found in conditional statements and loop statements where they are used to determine a condition. • In both conditional and loop statements, comparisons are found inside parentheses.
And	*comparison1* && *comparison2*	• Logical operator that combines two or more comparison expressions into one. • Both comparison expressions must be true for the overall expression to be true.	• Usually found in conditional statements and loop statements inside parentheses where they are used to combine multiple comparison statements.
Or	*comparison1* || *comparison2*	• Logical operator that combines two or more comparison expressions into one. • One or more comparison expressions must be true for the overall expression to be true.	

Command	Syntax	Purpose	Location in Sketch

```
  #include <PRIZM.h>                                      // include PRIZM library
  PRIZM prizm;                                            // instantiate a PRIZM object "prizm" so we can use its functions
  int obstacleDist = 25;                                                     // set the range for an obstacle at 25 centimeters
  int warningDist = 40;                                                      // set the range for a warning

void setup() {
  prizm.PrizmBegin();                                                        // initialize PRIZM
  prizm.setMotorInvert(1,1);                                                 // invert the direction of DC Motor 1
}
void loop() {
  if(prizm.readSonicSensorCM(4) > obstacleDist){                             // determine if obstacle is out of range
    if(prizm.readSonicSensorCM(4) < warningDist){prizm.setGreenLED(HIGH);} // if obstacle appears within warning range,...
                                                                           // ...turn on the green LED
    prizm.setMotorPowers(35,35);                                           // drive forward while obstacle is out of range
    prizm.setRedLED(LOW);                                                  // turn off the red LED
  }
  else {                                                                   // detected obstacle
    prizm.setGreenLED(LOW);                                                // turn off the green LED
    prizm.setRedLED(HIGH);                                                 // turn on the red LED
    prizm.setMotorPowers(125,125);                                         // stop the robot
    delay(500);                                                            // wait half a second
    prizm.setMotorPowers(35,-35);                                          // make a right turn
    delay(500);                                                            // turn for half a second
    prizm.setMotorPowers(125,125);                                         // stop the robot
  }
  if(prizm.readSonicSensorCM(4)<10 || prizm.readBatteryVoltage()<1000){    // if the robot is too close to an object or...
    prizm.PrizmEnd();                                                      // ...the battery is lower than 10 volts,...
  }                                                                        // ...end the program
}
```

This sketch uses conditional statements to keep a robot from driving into an obstacle. Two variables are declared at the beginning to set an obstacle distance and a warning distance. The main loop starts with an if-else statement to determine if an obstacle is out of range, and a nested if statement (the second if statement) turns the green LED on if an obstacle is within the warning distance. Continuing in the true part of the if-else statement, the robot will drive forward at 35% power with the red LED off.

The else part of the if-else statement is the false condition, meaning an obstacle has been detected within range. The LEDs are changed to indicate the obstacle is detected, and the robot performs a slight pivot turn.

The final if statement checks for two conditions using the **or** logical operator, and if either condition is true, the PRIZM ends, and the program is over. But as long as both conditions are false, the program repeats the main loop.

Command	Syntax	Purpose	Location in Sketch

Main loop	void loop() {
	}
		• Contains the section of code that runs repeatedly.
		• No conditions must be met for this loop to run.
			• Comes after the **void setup()** section.

While loop	while(*condition*){
	}
		• Repeats a series of commands while the condition inside the parentheses is met.
		• The condition is tested before the loop runs.
		• If the condition inside the while loop parentheses is false, the loop will not run.
		• While loops will repeat forever until something changes – either a tested variable in the condition or an external factor such as sensor data.
			• Most often used in the **void loop()** section of a sketch or in a called function.

Do-while loop	do{
	} while (*condition*);
		• Is similar to a while loop except the condition is checked at the end of the loop instead of the beginning.
		• Will repeat forever until the condition at the end of the loop is met.
		• Can be understood as "do these things and then, if a condition is true, do those things again."
		• Do-while loops always run at least one time.	

For loop	for (*initialization*; *condition*; *increment*) {
	}
		• Is used to repeat a set of commands a designated number of times.
		• Is often used to gradually change the status of an output device.
		• A counter is usually used to terminate the loop when a given condition for that counter is met.
		• For loops are composed of three statements: initialization, condition, and increment.
		◦ The initialization declares a variable for use in the loop as a certain type and sets its initial value.
		◦ The condition declares what must be true for the loop to repeat.
		◦ The increment changes the initialized variable so the condition can be checked again, and the loop is repeated or exited.	

Command Syntax Purpose Location in Sketch

```
  #include <PRIZM.h>                          // include PRIZM library
  PRIZM prizm;                                // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {
  prizm.PrizmBegin();                         // initialize PRIZM
  prizm.setMotorInvert(1,1);                  // invert the direction of DC Motor 1
}
void loop() {                                 // start the main loop
  for(int i = 1; i<=100; i++){                // this for loop will run 100 iterations with i increasing by 1 each iteration
    prizm.setMotorPowers(i,i);                // as i changes by 1 each iteration, the robot gradually accelerates
    delay(20);                                // the robot accelerates to 100% over a 2-second span (20 ms * 100 iterations)
  }                                           // after 100 iterations, the robot continues to drive forward at 100%

  while(prizm.readSonicSensorCM(3) > 25) {    // loop while an obstacle is greater than 25 cm away
    prizm.setGreenLED(HIGH);                  // turn on the green LED
    prizm.setRedLED(LOW);                     // turn off the red LED
    delay (40);                               // give the Ultrasonic Sensor time to receive its own signal
  }
  prizm.setMotorPowers(125,125);              // an obstacle is now within 25 cm, so stop the motors
  prizm.setGreenLED(LOW);                     // turn off the green LED
  prizm.setRedLED(HIGH);                      // turn on the red LED
  delay(1000);                                // wait 1 second before the next action

  do {                                        // start of a do-while loop
    prizm.setMotorPowers(25,-25);             // pivot the robot at 25% power away from obstacle
    prizm.setGreenLED(HIGH);                  // turn on the green LED
    prizm.setRedLED(HIGH);                    // turn on the red LED
  } while  (prizm.readSonicSensorCM(3) < 100); // repeat the do-while loop (continue turn) while obstacle is within 100 cm
  prizm.setMotorPowers(125,125);              // obstacle is now out of range, so stop the motors
  delay(1000);                                // wait 1 second
}                                             // repeat the main loop
```

This sketch utilizes four loops (main loop, for loop, while loop, and do-while loop) to keep a robot from crashing into an obstacle. The for loop gradually brings the robot up to speed as **i** increases by one each iteration of the loop up to 100% power. The while loop has the robot drive straight while the Ultrasonic Sensor reading is greater than 25 cm. When an obstacle is detected, the motors stop. The do-while loop pivots the robot until the obstacle is at least 100 cm away from the robot. When this happens, the robot stops and the main loop repeats, starting the entire process over again.

Command	Syntax	Purpose	Location in Sketch

Declare an integer variable	int *variable* = *value*;
	Examples:
	int minDist = 0;
	int maxDist = 600;
		• Declares an integer type variable and sets its value.
		• Variable names can be any letter/number combination that isn't used for another command, function, or value in the *Arduino IDE*.
		• Integer range is -32,768 and 32,768.
			• When and where a variable can be used depends on where the variable is declared in the sketch.
			• Global variables:
			◦ Are declared outside of a function (including the **void setup()** and **void loop()** functions).
			◦ Can be used anywhere in the sketch.
			◦ Are generally declared before the **void setup()** section.
			• Local variables:
			◦ Are declared inside the function they belong to.
			◦ Keeps other functions from inadvertently modifying variables used by another function.
			◦ Can be declared in a for loop and is used only in that for loop.

Declare a long variable	long *variable* = *value*;
	Examples:
	long speedOfLight = 186000L;
	long E = mass*speedOfLight*speedOfLight;
		• Declares a long type variable and sets its value.
		• Variable names can be any letter/number combination that isn't used for another command, function, or value in the *Arduino IDE*.
		• Long range is -2,147,483,648 to 2,147,483,647.
		• If doing math with integers, at least one of the numbers must be followed by an L, forcing it to be a long variable.	

Declare a floating-point variable	float *variable* = *value*; Examples: float x = 3.14 float circ = x*10.16	• Declares a decimal number with 6-7 decimal places of precision. • Variable names can be any letter/number combination that isn't used for another command, function, or value in the *Arduino IDE*. • Floating-point decimals are often used for math applications where decimals matter. However, because TETRIX applications utilize integer values, floating-point values must be converted to integers before they can be outputted to a device such as a motor, servo, or encoder.	

Set a variable	*variable* = *value*;
	Examples:
	dist = prizm.readSonicSensorCM(3);
	rate = 2.54;
	x = x + 1;
	voltage = prizm.readBatteryVoltage();
	circ = 2*3.14*rad;
		• Assigns a value to a variable.
		• A single equal sign assigns value, whereas a double equal sign compares one value to another.
		• Make sure that the value assigned is of the same type as the declared variable.
		◦ For example, if you have an integer variable **x** equal to 1 and an integer value **y** equal to 2, and you set an integer variable **z** equal to **x**/**y**, then even though 1/2 is equal to 0.5, **z** will be assigned a value of 0 because **z** is an integer variable.
			• Global variables:
			◦ Can happen anywhere in the sketch.
			◦ Are generally manipulated in the **void loop()** section or other functions.
			• Local variables:
			◦ Can happen only in the function they are declared in.

Command **Syntax** **Purpose** **Location in Sketch**

```
#include <PRIZM.h>                        // include the PRIZM library in the sketch
PRIZM prizm;                              // instantiate a PRIZM object “prizm” so we can use its functions
int wheelDiam = 4;                        // global variable: diameter of a 4" standard TETRIX wheel
float wheelCirc = wheelDiam*M_PI;         // global variable: circumference of the wheel as a decimal
long totalTime = 0;                       // global variable: total time in milliseconds for robot to run the course
int totalDist = 0;                        // global variable: total distance in inches robot travels

void setup() {
  prizm.PrizmBegin();                     // initialize the PRIZM controller
  prizm.setMotorInvert(2,1);              // invert Motor 2
}
void loop() {
  for (int x = 0; x<=720; x=x+5){         // x is a local variable only for this for loop that increments by 5 each iteration
    prizm.setMotorSpeeds(x,x);            // bring robot up to max speed gradually in increments of 5 deg/sec
  }                                       // end the for loop
  prizm.setGreenLED(HIGH);                // turn on the green LED
  prizm.resetEncoders();                  // reset the encoders
  long startTime = millis();              // local variable for this function: set startTime = to current PRIZM time in ms
  while(prizm.readLineSensor(3) == 0){}               // robot continues to drive forward until finish line is detected

  totalTime = round((millis() - startTime)/1000);     // calculate total time in milliseconds and convert to seconds
  totalDist = round(prizm.readEncoderDegrees(1) * wheelCirc / 360);     // calculate total distance in inches
  prizm.setMotorPowers(0,0);                                            // stop the motors
  displayVelocity();                                                    // run the function displayVelocity
  prizm.PrizmEnd();                                                     // end the program
}
void displayVelocity(){
  prizm.setGreenLED(LOW);                      // turn off the green LED
  delay(2000);                                 // wait 2 seconds
  int velocity = round(totalDist / totalTime); // declare local variable velocity as integer equal to totalDist ÷ totalTime
  for (int i = 0; i<=velocity; i++){           // declare i as local variable in this for loop only and set it equal to 0
    prizm.setRedLED(HIGH);                     // run loop velocity number of times by increasing i until i equals velocity
    delay (500);                               // for each loop iteration, flash the red LED to indicate the velocity
    prizm.setRedLED(LOW);
    delay(500);
  }
}
```

This sketch calculates the velocity of a robot in inches per second as it drives toward a finish line and then indicates the calculated velocity by flashing the red LED. The program uses both global and local variables to accomplish its tasks.

Command	Syntax	Purpose	Location in Sketch

Declare a called function	void *functionName*() {
	}
	functionName: any alpha-numeric combination that isn't an *Arduino IDE* command or variable name.
		• Allows user to create modular sections of code.
		• Is useful for defining a repetitive task so the code has to be written only one time.
		• Is useful for testing/troubleshooting code so specific segments of the code can be isolated.
		• Makes code more efficient, compact, and organized.
		• Variables used in a called function must be global variables or local variables for that function.
		• The **void setup()** and **void loop()** sections can be considered functions.
			• Called functions must be declared outside the brackets for the **void setup()** and **void loop()** sections.
			• They are commonly placed after the closing bracket for the **void loop()** section.

Call a function	*functionName*();	• Calls a function to perform a specific task. • Jumps out of the current location in the sketch to run the called function and then returns to where the function was called.	• Can be anywhere in a sketch including within another function.
Pass values to a called function	*functionName*(*value1*, *valueX*); *values*: can be variables of any type, or other data to pass	• Is used to pass variables or other information to a called function. • Values must be received by the called function when it is declared.	• Can be anywhere in a sketch including within another function.

Receive values for a called function	void *functionName*(*value1*, *valueX*) {
	}
		• Is used to receive values passed from another function/section for use within the function.	• Values are received in the parentheses of a called function.

Return	return *variableName*;	• Returns a value from a function to where the function was called.	• End of a called function.

```
void setup(){
  Serial.begin(9600);               // configure the serial monitor for 9600 baud rate
}
void loop() {
  int i = 2;                        // declare i as a local integer variable equal to 2 for use in the main loop
  int j = 3;                        // declare j as a local integer variable equal to 3 for use in the main loop
  int k = 0;                        // declare k as a local integer variable equal to 0 for use in the main loop
  k = multiplyNumbers(i, j);        // k will equal 6 as i(2) and j(3) are passed to the multiplyNumbers function
  Serial.println(k);                // print k in the serial monitor; a value of 6 will be displayed
}
int multiplyNumbers(int x, int y){  // receive i and j as local integer variables x and y so they can be used in the function
  int result = x * y;               // declare result as a local variable equal to x*y, which is 6
  return result;                    // passes the variable result (6) back to the command where the function was called
}
```

This sketch uses a called function to multiply two variables that are passed into it, and then it returns and displays the product of the two variables.

Command	Syntax	Purpose	Location in Sketch

```
  #include <PRIZM.h>          // include PRIZM library
  PRIZM prizm;                // instantiate a PRIZM object "prizm" so we can use its functions
  int randNumb = 0;           // declare an integer variable for a random number
  int mPower = 25;            // declare a motor power setting
void setup() {
  prizm.PrizmBegin();         // initialize PRIZM
  prizm.setMotorInvert(1,1);  // invert the direction of DC Motor 1 to harmonize direction
}
void loop() {
  if (prizm.readSonicSensorCM(2)<=100){        // determine if an obstacle is within 100 cm
    aboutTurn();                                // if there is an obstacle, call the aboutTurn function
  }
  randNumb = random(1,4);                       // pick a random number between 1 and 4
  if(randNumb == 1){leftTurn();}                // if the random number is 1, then call the leftTurn function
  else if(randNumb == 2){forward();}            // if the random number is 2, then call the forward function
  else if (randNumb == 3){rightTurn();}         // if the random number is 3, then call the rightTurn function
  else {aboutTurn();}                           // if the random number is 4, then call the aboutTurn function
  while (prizm.readSonicSensorCM(2)>=100){      // determine if an obstacle is within 100 cm
    forward();                                  // if there is no obstacle, call the forward function
  }

}                                               // repeat the main loop
void forward(){                                 // function to go forward
  prizm.setMotorPowers(mPower,mPower);          // go forward at designated motor power
  delay(1000);                                  // go forward for 1 second
}
void rightTurn(){                               // function for a right turn
  prizm.setMotorPowers(mPower,-mPower);         // make a right turn at designated motor power
  delay(1000);                                  // turn right for 1 second
}
void leftTurn(){                                // function for a left turn
  prizm.setMotorPowers(-mPower,mPower);         // make a left turn at designated motor power
  delay(1000);                                  // turn left for 1 second
}
void aboutTurn(){                               // function for an about turn
  prizm.setMotorPowers(-mPower,mPower);         // turn around at designated motor power
  delay(1700);                                  // turn for 1.7 seconds
}
```

This sketch randomly chooses a direction and as long as there are no obstacles in that direction, the robot will move that direction. There are four called functions. Two of the functions (**aboutTurn** and **forward**) are called from two different locations in the main loop.

Command	Syntax	Purpose	Location in Sketch
Delay in milliseconds	delay(*milliseconds*);	• Pauses the program for designated time. • Can be used to pause the program while another task completes. • 1 second equals 1,000 milliseconds.	• Can be used anywhere in the sketch.
Current time index	millis()	• Recalls the number of milliseconds that have expired since the program began running. • Can be used to determine the time between two events • Can be used to time events/actions without pausing the program like a delay does. It allows input or output actions to occur during the designated time interval.	• Can be used anywhere in the sketch.

```
  #include <PRIZM.h>          // include the PRIZM library
  PRIZM prizm;                // instantiate a PRIZM object "prizm" so we can use its functions
  long msInDay = 86400000;    // 86400000 milliseconds in a day
  long msInHour = 3600000;    // 3600000 milliseconds in an hour
  long msInMinute = 60000;    // 60000 milliseconds in a minute
  long msInSecond =  1000;    // 1000 milliseconds in a second
void setup() {
   prizm.PrizmBegin();        // initialize the PRIZM controller
   Serial.begin(9600);        // configure the serial monitor for 9600 baud rate
}
void loop() {
  long timeNow = millis();                                  // get the current millisecond time from PRIZM
  int days = timeNow / msInDay ;                            // calculate number of days
  int hours = (timeNow % msInDay) / msInHour;               // calculate hours from remainder of days division
  int minutes = ((timeNow % msInDay) % msInHour) / msInMinute ; // calculate minutes from remainder of hours division
  int seconds = (((timeNow % msInDay) % msInHour) % msInMinute) // calculate seconds from remainder of minutes division
         / msInSecond;                                      // continue calculation from previous line
  Serial.print(days);                                       // display the days
  printDigits(hours);                                       // pass hours to called function to display hours
  printDigits(minutes);                                     // pass minutes to called function to display minutes
  printDigits(seconds);                                     // pass seconds to called function to display seconds
  Serial.println();                                         // add a line break to the display
  delay(1000);                                              // delay 1 second before repeating loop to display next time
}
void printDigits(int digits){          // called function to display time by receiving value as integer named digits
 Serial.print(":");                    // add : to the display line to separate units of time
 if(digits < 10){Serial.print('0');}   // add a 0 after colon to display in 0:00:00:00 format instead of 0:0:0:0 format
 Serial.print(digits);                 // add the variable digits to the display line
}
```

This sketch uses the **millis()** command to display the days, hours, minutes, and seconds since the program started running in a 0:00:00:00 format.

Command	Syntax	Purpose	Location in Sketch
Set up serial monitor	Serial.begin(*baud*); *baud*: the data rate in bits per second for data transmission	• Configures the serial monitor so it can be used. • Is used to communicate between PRIZM and a computer or other device. • A typical baud rate for communicating between PRIZM and a computer is 9600. Faster baud rates are possible but not necessary.	• Most often found in the **void setup()** section of a sketch.
Display information in the serial monitor	Serial.print();	• Displays data, variables, sensor data, and text in the serial monitor associated with the *Arduino IDE*. • Is useful for debugging code, monitoring sensor data, checking calculations, and so on. • Information to display should be contained inside the parentheses. • Text-based information needs to be surrounded by quotes.	• Can be used anywhere after the **Serial.begin()** command.
Add a new line in the serial monitor	Serial.println();	• Starts a new line in the serial monitor after displaying what is in parentheses.	• Can be used anywhere after the **Serial.begin()** command.

```
  #include <PRIZM.h>      // include the PRIZM library
  PRIZM prizm;            // instantiate a PRIZM object “prizm” so we can use its functions
void setup() {
  prizm.PrizmBegin();     // initialize PRIZM
  Serial.begin(9600);     // configure the serial monitor for 9600 baud rate
 }
void loop() {                                   // start the main loop
  Serial.print(prizm.readSonicSensorCM(2));     // print the CM distance to the serial monitor
  Serial.println(" Centimeters");               // print " Centimeters" and add a line break after
  delay(200);                                   // slow down loop so Ultrasonic Sensor can receive data
}
```

This sketch reads the data from the Ultrasonic Sensor and outputs it to the serial monitor. Data is outputted in the form of ## Centimeters every 200 milliseconds.

TETRIX® PRIZM® and *Arduino IDE* Reference Guide

Understanding common PRIZM commands and functions in the *Arduino Software (IDE)*

Call Toll-Free
800•835•0686

Visit Us Online at
Pitsco.com