TETRIX® PRIZM™ ESP32 – Arduino Library API Reference (PROTOTYPE)

Description	Function	Coding Example
Prizm Begin Is called in the Arduino code setup() loop. Initializes the PRIZM controller.	PrizmBegin**();** Data Type: None	PrizmBegin(); *Reset and initialize PRIZM controller.*
Prizm End When called, immediately terminates a program and resets the PRIZM controller.	PrizmEnd**();** Data Type: None	PrizmEnd(); *Terminate a PRIZM program and reset controller.*
Set Green LED Sets the PRIZM green indicator LED to on or off.	setGreenLED(state); Data Type: *state* = integer Data Range: *state* = 1 or 0 or *state* = HIGH or LOW	setGreenLED(HIGH); *or* setGreenLED(1); *Turn green LED on.* setGreenLED(LOW); *or* setGreenLED(0); *Turn green LED off.*
Set DC Motor Power Sets the power level and direction of a TETRIX DC Motor connected to the PRIZM DC motor ports. Power level range is 0 to 100. Direction is set by the sign (+/-) of the power level. Power level 0 = stop in coast mode. Power level 125 = stop in brake mode.	setMotorPower(motor#, power); Data Type: *motor#* = integer *power* = integer Data Range: *motor#* = 1, 2, 3, 4 *power* = -100 to 100 or *power* = 125 (brake mode) *power* = 0 (stop in coast mode)	setMotorPower(1, 50); *Spin Motor 1 clockwise at 50% power.* setMotorPower(2, -50%); *Spin Motor 2 counterclockwise at 50% power.* setMotorPower(1, 0); *Turn off Motor 1 in coast mode.* setMotorPower(2, 125); *Turn off Motor 2 in brake mode.*
Set DC Motor Powers Simultaneously sets the power level and direction of **ALL** TETRIX DC Motors connected to the PRIZM motor ports. All four PRIZM DC Motor channel parameters are set with a single statement. The power level range is 0 to 100. Direction is set by the sign (+/-) of the power level. Power level 0 = stop in coast mode. Power level 125 = stop in brake mode.	setMotorPowers(power1, power2, power3, power4); Data Type: *power(#)* = integer Data Range: *power(#)* = -100 to 100 or *power(#)* = 125 (brake mode) *power(#)* = 0 (coast mode)	setMotorPowers(50, 50, 50, 50); *Spin Motors 1 – 4 clockwise at 50% power.* setMotorPowers(-50, 50, -50, 50); *Spin Motors1 and 3 counterclockwise and Motors 2 and 4 clockwise at 50% power.* setMotorPowers(0, 0, 0, 0); *Turn off all motors in coast mode.* setMotorPowers(125, 125); *Turn off all motors in brake mode.*
Set DC Motor Speed Uses velocity PID control to set the constant speed of a TETRIX DC Motor with quadrature encoder. The *speed* parameter range is 0 to 720 degrees per second (DPS). The sign (+/-) of the *speed* parameter controls direction of rotation.	setMotorSpeed(motor#, speed); Data Type: *motor#* = integer *speed* = integer Data Range: *motor#* = 1,2,3,4, *speed* = -720 to 720	setMotorSpeed(1, 360); *Spin Motor 1 clockwise at a constant speed of 360 DPS.* setMotorSpeed(1, -360); *Spin Motor 1 counterclockwise at a constant speed of 360 DPS.*
Set DC Motor Speeds Uses velocity PID control to simultaneously set the constant speeds of **both** TETRIX DC Motor channels with quadrature encoders. All DC motor channel parameters are set with a single statement. The *speed* parameter range is 0 to 720 degrees per second (DPS). The sign (+/-) of the *speed* parameter controls direction of rotation.	setMotorSpeeds(speed1, speed2, speed3, speed4); Data Type: *speed(#)* = integer Data Range: *speed(#)* = -720 to 720	setMotorSpeeds(360, 360, 360, 360); *Spin Motors 1-4 clockwise at a constant speed of 360 DPS.* setMotorSpeeds(360, -360, 360, -360); *Spin Motor 1 clockwise, Motor 2 counterclockwise, Motor 3 clockwise, Motor 4 counterclockwise at a constant speed of 360 DPS.* setMotorSpeeds(360, -180, 0, 0); *Spin Motor 1 clockwise and Motor 2 counterclockwise at a constant speed of 180 DPS. Set Motors 3 and 4 to off.*

Set DC Motor Target Implements velocity and positional PID control to set the constant speed and the encoder count target holding position of a TETRIX DC Motor and quadrature encoder. The *speed* parameter range is 0 to 720 degrees per second (DPS). The encoder count target position is a signed long integer from -2,147,483,648 to 2,147,483,647. Each encoder count = 1/4-degree resolution.	setMotorTarget(motor#, speed, target); Data Type: *motor#* = integer *speed* = integer *target* = long Data Range: *motor#* = 1, 2, 3, 4 *speed* = 0 to 720 *target* = -2147483648 to 2147483647	setMotorTarget(1, 360, 1440); *Spin Motor 1 at a constant speed of 360 DPS until encoder 1 count equals 1,440. When at encoder target count, hold position in a servo-like mode.* setMotorTarget(2, 180, -1440); *Spin Motor 2 at a constant speed of 180 DPS until encoder 2 count equals -1,440 (1 revolution). When at encoder target count, hold position in a servo-like mode.*
Set DC Motor Targets Implements velocity and positional PID control to simultaneously set the constant speeds and the encoder count target holding positions of **ALL** TETRIX DC Motor channels utilizing quadrature encoders. All motor channel parameters are set with a single statement. The speed parameter range is 0 to 720 degrees per second (DPS). The encoder count target position is a signed long integer from -2,147,483,648 to 2,147,483,647. Each encoder count = 1/4-degree resolution.	setMotorTargets(speed1, target1, speed2, target2, speed3, target3, speed4, target4); Data Type: *speed(#)* = integer *target(#)* = long Data Range: *speed(#)* = 0 to 720 *target(#)* = -2147483648 to 2147483647	setMotorTargets(360, 1440, 360, 1440, 360, 1440, 360, 1440); *Spin Motors 1-4 at a constant speed of 360 DPS until each motor encoder count equals 1,440. When a motor reaches its encoder target count, hold position in a servo-like mode.* setMotorTargets(360, 1440, 180, 2880, 360, 1440, 180, 720); *Spin Motor 1 and 3 at a constant speed of 360 DPS until encoder count equals 1,440. Spin Motor 2 at a constant speed of 180 DPS until encoder 2 equals 2,880. Spin Motor 4 at a constant speed of 180 DPS until encoder 4 equals 720. Each motor will hold its position in a servo-like mode when it reaches the encoder target.* ***Note:*** *One encoder count equals 1/4-degree resolution. For example, 1 motor revolution equals 1,440 encoder counts (1,440 / 4 = 360).*
Set Motor Degree Implements velocity and positional PID control to set the constant speed and the degree target holding position of a TETRIX DC Motor with quadrature encoder. The speed parameter range is 0 to 720 degrees per second (DPS). The encoder degrees target position is a signed long integer from -536,870,912 to 536,870,911 with a 1-degree resolution.	setMotorDegree(motor#, speed, degrees); Data Type: *motor#* = integer *speed* = integer *degrees* = long Data Range: *motor#* = 1 or 2 *speed* = 0 to 720 *degrees* = -536870912 to 536870911	setMotorDegree(1, 180, 360); *Spin Motor 1 at a constant speed of 180 DPS until encoder 1 degree count equals 360. When at encoder target degree count, hold position in a servo-like mode.* setMotorDegree(2, 90, 180); *Spin Motor 2 at a constant speed of 90 DPS until encoder 2 degree count equals 180. When at encoder target degree count, hold position in a servo-like mode.*
Set Motor Degrees Implements velocity and positional PID control to set the constant speeds and the degree target holding positions of **ALL** TETRIX DC Motor channels with quadrature encoders. All motor channel parameters are set with a single statement. The speed parameter range is 0 to 720 degrees per second (DPS). The encoder degree target position is a signed long integer from -536,870,912 to 536,870,911 with a 1-degree resolution.	setMotorDegrees(speed1, degrees1, speed2, degrees2, speed3, degrees3, speed4, degrees4); Data Type: *speed(#)* = integer *degrees(#)* = long Data Range: *speed(#)* = 0 to 720 *degrees(#)* = -536870912 to 536870911	setMotorDegrees(180, 360, 180, 360, 180, 360, 180, 360); *Spin Motor 1-4 at a constant speed of 180 DPS until each motor encoder degree count equals 360. When a motor reaches its degree target count, hold position in a servo-like mode.* setMotorDegrees(360, 720, 90, 360, 50, 180, 100, 720); *Spin Motor 1 at a constant speed of 360 DPS until encoder 1 degree count equals 720. Spin Motor 2 at a constant speed of 90 DPS until encoder 2 degree equals 360. Spin Motor 3 at a constant speed of 50 DPS until encoder 3 count equals 180. Spin Motor 4 at a constant speed of 100 DPS until encoder 4 count equals 720.* *Each motor will hold its position in a servo-like mode when it reaches the encoder target.*
Set Motor Direction Invert Inverts the forward/reverse direction mapping of a DC motor channel. This function is intended to harmonize the forward and reverse directions for motors on opposite sides of a skid-steer robot chassis. Inverting one motor channel can make coding opposite-facing DC motors working in tandem more intuitive. An *invert* parameter of 1 = invert. An *invert* parameter of 0 = no invert. The default is no invert.	setMotorInvert(motor#, invert); Data Type: *motor#* = integer *invert* = boolean Data Range: *motor#* = 1 or 2 *invert* = 0 or 1, TRUE or FALSE	setMotorInvert(1, 1); *Invert the spin direction mapping of Motor 1.* setMotorInvert(2, 1); *Invert the spin direction mapping of Motor 2.* setMotorInvert(1, 0); *Do not invert the spin direction mapping of Motor 1.* setMotorInvert(2, 0); *Do not invert the spin direction mapping of Motor 2.*

Read Motor Busy Status The busy flag can be read to check on the status of a DC motor that is operating in position target PID mode. The motor busy status will return "1" if it is moving toward a positional target (degrees or encoder count). When it has reached its target and is in hold mode, the busy status will return "0."	readMotorBusy(motor#); Data Type: *motor#* = integer Data Range: *motor#* = 1, 2, 3, 4 Data Type Returned: *value* = boolean	readMotorBusy(1); *Return the busy status of Motor 1.* readMotorBusy(2); *Return the busy status of Motor 2.* readMotorBusy(3); *Return the busy status of Motor 3.* readMotorBusy(4); *Return the busy status of Motor 4.*
Read Encoder Count Reads the encoder count value. The PRIZM controller uses encoder pulse data to implement PID control of a TETRIX DC Motor connected to the motor ports. The PRIZM controller counts the number of pulses produced over a set time period to accurately control velocity and position. Each 1/4 degree equals one pulse, or count, or 1 degree of rotation equals 4 encoder counts. The current count can be read to determine a motor's shaft position. The total count accumulation can range from -2,147,483,648 to 2,147,483,647. A clockwise rotation adds to the count value, while a counterclockwise rotation subtracts from the count value. The encoder values are set to 0 at power-up and reset.	readEncoderCount(enc#); Data Type: *enc#* = integer Data Range: *enc#* = 1, 2, 3, 4 Data Type Returned: *value* = long	readEncoderCount(1); *Read the current count value of encoder 1 (ENC 1 port).* readEncoderCount(2); *Read the current count value of encoder 2 (ENC 2 port).* readEncoderCount(3); *Read the current count value of encoder 3 (ENC 3 port).* readEncoderCount(4); *Read the current count value of encoder 4 (ENC 4 port).*
Read Encoder Degrees Reads the encoder degree value. The PRIZM controller uses encoder pulse data to implement PID control of a TETRIX DC Motor connected to the motor ports. The PRIZM controller counts the number of pulses produced over a set time period to accurately control velocity and position. This function is similar to the encoder count function, but instead of returning the raw encoder count value, it returns the motor shaft position in degrees. The total degree count accumulation can range from -536,870,912 to 536,870,911. A clockwise rotation adds to the count value, while a counterclockwise rotation subtracts from the count value. The encoder values are set to 0 at power-up and reset.	readEncoderDegrees(enc#); Data Type: *enc#* = integer Data Range: *enc#* = 1, 2, 3, 4 Data Type Returned: *value* = long	readEncoderDegrees(1); *Read the current degree count value of encoder 1 (ENC 1 port).* readEncoderDegrees(2); *Read the current degree count value of encoder 2 (ENC 2 port).* readEncoderDegrees(3); *Read the current degree count value of encoder 3 (ENC 3 port).* readEncoderDegrees(4); *Read the current degree count value of encoder 4 (ENC 4 port).*
Reset a specific Encoder Calling this function will reset the encoder count accumulator to 0.	resetEncoder(enc#); Data Type: *enc#* = integer Data Range: *enc#* = 1, 2, 3, 4	resetEncoder(1); *Reset the encoder 1 count to 0.* resetEncoder(2); *Reset the encoder 2 count to 0.* resetEncoder(3); *Reset the encoder 3 count to 0.* resetEncoder(4); *Reset the encoder 4 count to 0.*
Reset All Encoders (1, 2, 3 4) Calling this function will reset **ALL** encoders (1-4) count accumulators to 0.	resetEncoders(); Data Type: None	resetEncoders(); *Reset the encoders 1,2,3,4 counts to 0.*
Read Line Sensor Output Reads the digital output of the Line Finder Sensor connected to a PRIZM sensor port. The value read is "0" when reflected light is received (detecting a light-colored surface) and "1" when light is not received (detecting a dark-colored surface, such as a line).	readLineSensor(port#); Data Type: *port#* = integer Data Range: *port#* GPIO port# 1-5 Data Type Returned: *value* = boolean (0 or 1)	readLineSensor(2); *Read the digital value of a Line Finder Sensor on GPIO port 2.*

Read Ultrasonic Sensor in Centimeters Reads the distance in centimeters of an object placed in front of the Ultrasonic Sensor. The sensor is modulated at 42 kHz and has a range of 3 to 400 centimeters. The value read is an integer.	readSonicSensorCM(port#); Data Type: *port#* = integer Data Range: *port#* GPIO port# 2-5 Data Type Returned: *value* = integer (3 to 400) Min and max might slightly vary.	readSonicSensorCM(3); *Read the distance in centimeters of an object placed in front of the Ultrasonic Sensor connected to GPIO port 3.*
Read Ultrasonic Sensor in Inches Reads the distance in inches of an object placed in front of the Ultrasonic Sensor. The sensor is modulated at 42 kHz and has a range of 2 to 150 inches. The value read is an integer.	readSonicSensorIN(port#); Data Type: *port#* = integer Data Range: *port#* GPIO port# 1-5 Data Type Returned: *value* = integer (2 to 150) Min and max might slightly vary.	readSonicSensorIN(4); *Read the distance in inches of an object placed in front of the Ultrasonic Sensor connected to GPIO port 4.*
Read Battery Pack Voltage Reads the voltage of the TETRIX battery pack powering the PRIZM controller. The value read is an integer.	readBatteryVoltage(); Data Type: None Data Type Returned: *value* = integer	readBatteryVoltage(); *Read the voltage of the TETRIX battery pack powering the PRIZM controller.* *Example: A value of 918 equals 9.18 volts.*
Read Green Button State Reads the state of the green PRIZM Start button. A returned value of "1" indicates a pressed state. A returned value of "0" indicates a not-pressed state.	readGreenButton(); Data Type: None Data Type Returned: *value* = boolean (0 or 1)	readGreenButton(); *Read the Green button. A value of 1 means button is pressed. A value of 0 means button is not pressed.*
Read Black Button State Reads the state of the green PRIZM Start button. A returned value of "1" indicates a pressed state. A returned value of "0" indicates a not-pressed state.	readBlackButton(); Data Type: None Data Type Returned: *value* = boolean (0 or 1)	readBlackButton(); *Read the Black button. A value of 1 means button is pressed. A value of 0 means button is not pressed.*
Set Speed of a Servo Motor This function sets the speed of a servo motor connected to a PRIZM servo port 1-6. The speed parameter can be 0 to 100%. The servo motor channel parameter can be any number 1 to 6. If not specified, the speed of a servo defaults to 100 (maximum speed). When a servo speed has been set, it will always move at the set speed until changed. Unless we are changing speeds, it will need to be called only once at the beginning of a program.	setServoSpeed(servo#, speed); Data Type: *servo#* = integer *speed* = integer Data Range: *servo#* = 1 to 6 *speed* = 0 to 100	setServoSpeed(1, 25); *Set the speed of servo channel 1 to 25%.* setServoSpeed(2, 50); *Set the speed of servo channel 2 to 50%.*
Set Speeds of All Servo Motors This function will set the speeds of all six servo channels simultaneously with a single command. All six speeds are in sequential order and can be 0 to 100%. All six servo speeds may be the same or different.	setServoSpeeds(speed1, speed2, speed3, speed4, speed5, speed6); Data Type: *speed1-speed6* = integer Data Range: *speed1-speed6* = 0 to 100	setServoSpeeds(25, 25, 25, 25, 25, 25); *Set the speeds of all six servo channels to 25%.* setServoSpeeds(25, 35, 45, 55, 65, 75); *Servo 1 speed = 25%* *Servo 2 speed = 35%* *Servo 3 speed = 45%* *Servo 4 speed = 55%* *Servo 5 speed = 65%* *Servo 6 speed = 75%*

Set Position of a Servo Motor Set the angular position of a servo motor connected to a PRIZM servo motor port 1-6. The *position* parameter can be any value between 0 and 180 degrees. Any value outside this range is ignored. Not all servos are the same, so be careful when operating a servo motor at the extreme ranges. Listen closely; if a servo is buzzing, it is pressing against its mechanical stop, which might damage the motor. If this happens, limit the range to values slightly greater than 0 and slightly less than 180 to avoid damage to the servo motor.	setServoPosition(servo#, position); Data Type: *servo#* = integer *position* = integer Data Range: *servo#* = 1 to 6 *position* = 0 to 180	setServoPosition(1, 90); *Set the angular position of Servo Motor 1 to 90 degrees.* setServoPosition(2, 130); *Set the angular position of Servo Motor 2 to 130 degrees.*
Set Positions of All Servo Motors Set the angular positions of all six servo motors connected to the PRIZM servo motor ports 1-6. The *position* parameter can be any value between 0 and 180 degrees. Any value outside this range is ignored. Not all servos are the same, so be careful when operating a servo motor at the extreme ranges. Listen closely; if a servo is buzzing, it is pressing against its mechanical stop, which might damage the motor. If this happens, limit the range to values slightly greater than 0 and slightly less than 180 to avoid damage to the servo motor.	setServoPositions(position1, position2, position3, position4, position5, position6); Data Type: *position1-position6* = integer Data Range: *position1-position6* = 0 to 180	setServoPositions(90, 90, 90, 90, 90, 90); *Set the angular positions of all six servo motors connected to PRIZM servo ports 1-6 to 90 degrees.* setServoPositions(10, 20, 30, 40, 50, 60); *Set the angular positions of all six servo motors connected to PRIZM servo ports 1-6.* *Servo 1 position = 10 degrees* *Servo 2 position = 20 degrees* *Servo 3 position = 30 degrees* *Servo 4 position = 40 degrees* *Servo 5 position = 50 degrees* *Servo 6 position = 60 degrees*
Read a Servo Position Reads the most recent commanded position of a servo motor connected to PRIZM servo ports 1-6. The value returned will be 0-180.	readServoPosition(servo#); Data Type: *servo#* = integer Data Range: *servo#* = 1 to 6 Data Type Returned: *value* = integer (0 to 180)	readServoPosition(1); *Read the most recent commanded position of Servo 1.* readServoPosition(2); *Read the most recent commanded position of Servo 2.*
Play a Tone Frequency Sets a tone frequency in (Hz) to the PRIZM sound speaker.	playTone(frequency); Data Type: *frequency* = integer Data Range: *frequency = 31 - 4978*	playTone(1000); *Play a 1Khz tone.*
Play a Musical Note Sets a note from the musical scale to the PRIZM sound speaker.	playNote(note, octave, duration); Data Type: *note* = char *octave* = integer *duration* = integer Data Range: *note = A – G* *octave = 1 – 8* *duration = milliseconds*	playNote(C, 3, 1000); *Play musical note C, octave 3 for 1 second)*
Set LED Pixel Color Data There is a three-pixel WS2812 addressable RGB LED array on-board the controller. This function sets the Red, Green, Blue (RGB) data value of an LED pixel to be displayed by the showLedData() function.	setLedColorData(pixel, r, g, b); Data Type: *pixel* = byte *r = byte* *g = byte* *b = byte* Data Range: *pixel = 1, 2, 3* *r,g,b = 0 - 255*	setLedColorData(1, 255, 0, 0); *Set the color data of pixel 1 to Red.* setLedColorData(2, 0, 255, 0); *Set the color data of pixel 2 to Green.* setLedColorData(3, 0, 0, 255); *Set the color data of pixel 3 to Blue.* showLedData(); *Send data to the LED pixel array.*

TETRIX® PRIZM™ ESP32 – Arduino Library API Reference (PROTOTYPE)

Set LED Color Data for All Pixels There is a three-pixel WS2812 addressable RGB LED array on-board the controller. This function sets the Red, Green, Blue (RGB) value of all LED pixels to be displayed by the showLedData() function.	setLedColorsData(r, g, b); Data Type: *r = byte* *g = byte* *b = byte* Data Range: *r,g,b = 0 - 255*	setLedColorsData(0, 0, 255); *Set the color of all three LED pixels to Blue.* showLedData(); *Send data to the LED pixel array.*
Show LED Color Data for All Pixels There is a three-pixel WS2812 addressable RGB LED array on-board the controller. This function sends the set color data values to the 3-pixel LED array for display.	showLedData();	setLedColorsData(0, 0, 255); *Set the color of all three LED pixels to Blue.* showLedData(); *Send data to the LED pixel array.*
Set LED pixel Color There is a three-pixel WS2812 addressable RGB LED array on-board the controller. This function immediately sets the Red, Green, Blue (RGB) color of an LED pixel.	setLedColor(pixel, r, g, b); Data Type: *pixel* = byte *r = byte* *g = byte* *b = byte* Data Range: *pixel = 1, 2, 3* *r,g,b = 0 - 255*	setLedColor(2, 255, 255, 0); *Set the color of LED pixel 2 to yellow.*
Set All LED pixel Colors There is a three-pixel WS2812 addressable RGB LED array on-board the controller. This function immediately sets the Red, Green, Blue (RGB) color of the three-pixel LED array.	setLedColors(r, g, b); Data Type: *r = byte* *g = byte* *b = byte* Data Range: *r,g,b = 0 - 255*	setLedColors(0, 255, 0); *Set the color of all three LED pixels to Green.*
Return a value from the Color Wheel This function returns a value from the color wheel which can be read into a variable and sent the RGB pixel array.	colorWheel(value); Data Type: *Value = byte* Data Range: *0-255*	pixel1 = colorWheel(128); pixel2 = colorWheel(80); pixel3 = colorWheel(255); *set each pixel variable to a color from the color wheel* setLedColors(pixel1, pixel2, pixel3); *display each pixel color*

Common ESP32-Arduino API also supported.

Link to the ESP32 Arduino Documentation: https://docs.espressif.com/projects/arduino-esp32/en/latest/getting_started.html

MicroPython Module for TETRIX Controller is in process.
The TETRIX controller will use the MicroPython firmware port for ESP-32.

Link to MicroPython documentation and firmware: https://micropython.org/