MATLAB® Support Package for Arduino® Hardware

User’s Guide

R2014a

MATLAB®

MathWorks®

MATLAB® Support Package for Arduino® Hardware User's Guide

Revision History

September 2014	Online only	New for Version 14.1.0 (R2014a)

Contents

Setup and Configuration

Install Support for Arduino Hardware

Install the MATLAB® Support Package for Arduino® Hardware to add support for Arduino hardware.

The installation adds these items on your host computer:

- Third-party software development tools
- MATLAB commands
- Examples

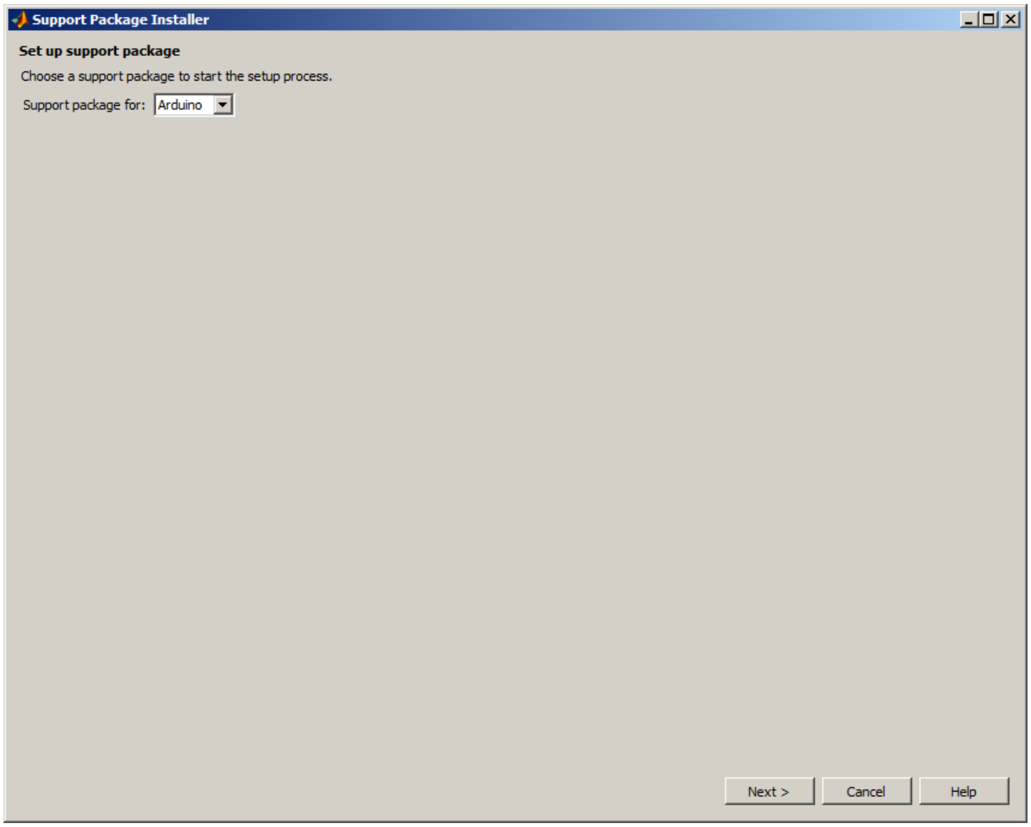
When you complete installation, you can use MATLAB commands to control, and retrieve data from, Arduino hardware and peripherals.

Install the Support Package

This example shows how to add support for Arduino hardware to the MATLAB product. After you complete this process, you can run MATLAB code on your Arduino hardware.

To install support for Arduino hardware:

1. In a MATLAB Command window, enter `supportPackageInstaller` to open the Support Package Installer.
2. Follow the instructions and default settings provided by Support Package Installer to complete the installation. For more information about the options on a particular screen, click the **Help** button.
3. When the installation is complete Windows users may see a Setup Support Package screen.

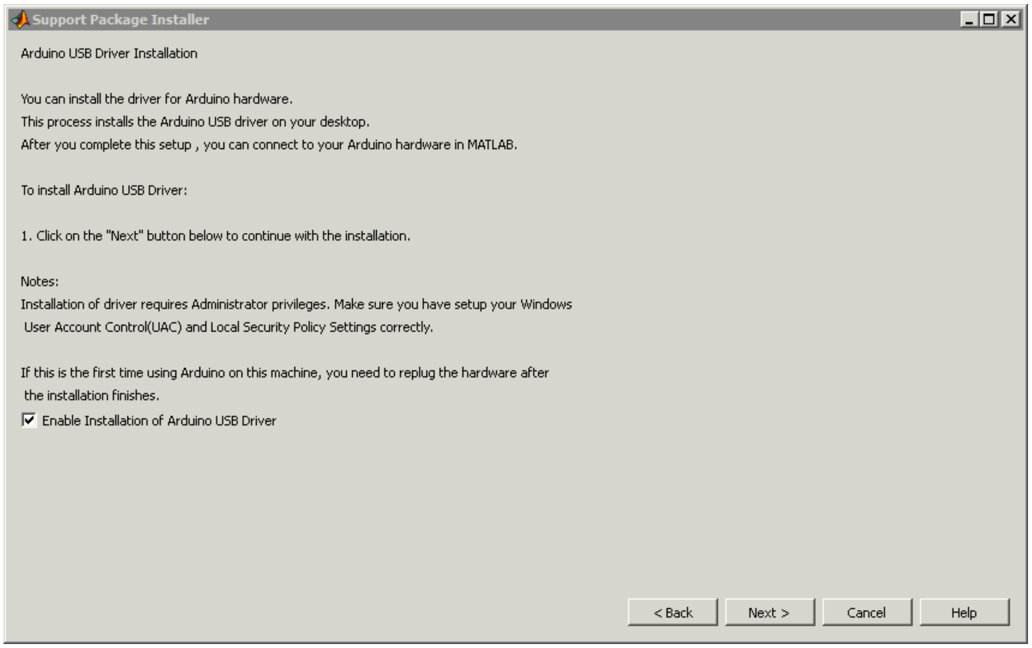


Set up support package

Choose a support package to start the setup process.

Support package for: Arduino

4 Click next to set up the support package. The installer prompts you for an Arduino Driver Installation.



Arduino USB Driver Installation

You can install the driver for Arduino hardware.
This process installs the Arduino USB driver on your desktop.
After you complete this setup , you can connect to your Arduino hardware in MATLAB.

To install Arduino USB Driver:

1. Click on the "Next" button below to continue with the installation.

Notes:
Installation of driver requires Administrator privileges. Make sure you have setup your Windows User Account Control(UAC) and Local Security Policy Settings correctly.

If this is the first time using Arduino on this machine, you need to replug the hardware after the installation finishes.

☑ Enable Installation of Arduino USB Driver

5 If you have administrative privileges, select **Enable Installation of Arduino USB Driver** and click next.

The installation process adds the following items to your host computer:

- Third-party software development tools, such as the Arduino software
- Additional driver and firmware.
- Examples for getting started and learning about specific features

---

Note: If you do not have administrative privileges or do not wish to update your drivers, you can skip this step and update later using the targetupdater command.

To update, repeat these steps.

Accessing Examples

To access featured examples for MATLAB Support Package for Arduino Hardware open MATLAB and type:

```
arduinoExamples
```

Arduino Hardware

Arduino is an open-source hardware available at a low cost. The design is based on a simple microcontroller and you can read and write to its physical computing platform. You can use both official Arduino hardware as well as hardware made by other vendors. Currently you can use the Arduino hardware manufactured by SparkFun Electronics. The Arduino interface connects to external devices like sensors and motors that you can communicate with using MATLAB.

MATLAB Support Package for Arduino Hardware enables you to communicate with Arduino devices via MATLAB command line interface. You can communicate with the device input and output peripherals and communication interface. You can also communicate sensors connected to Arduino hardware and actuate devices attached to Arduino hardware.

Read Write Functions

`configureDigitalPin` `readDigitalPin` `writeDigitalPin` `writePWMVoltage`

`writePWMDutyCycle` `playTone` `configureAnalogPin` `readVoltage` `arduinoExamples`

Supported Hardware

For a list of currently supported Arduino hardware, see the Supported Hardware page:

http://www.mathworks.com/hardware-support/arduino-matlab.html

Connect to Arduino Hardware

Make sure the Arduino is connected to the computer. If the device is unofficial, note the port and board name.

Connect to an official Arduino.

```
a = arduino
```

```
a = 

  arduino with properties:

                    Port: 'COM25'
                   Board: 'Uno'
     AvailableAnalogPins: [0 1 2 3 4 5]
    AvailableDigitalPins: [2 3 4 5 6 7 8 9 10 11 12 13]
               Libraries: {'I2C'  'SPI'  'Servo'}
```

Connect to an unofficial (clone) Arduino by specifying the port and board name.

```
a = arduino('com3','uno')
```

```
a = 

  arduino with properties:

                    Port: 'COM3'
                   Board: 'Uno'
     AvailableAnalogPins: [0 1 2 3 4 5]
     AvailableDigitalPins: [2 3 4 5 6 7 8 9 10 11 12 13]
               Libraries: {'I2C'  'SPI'  'Servo'}
```

Read and Write to Digital Pin

This example shows how to configure a push button on a digital pin on an Arduino hardware and read the state

Connect a push button to digital pin 12 on your Arduino.

Create an Arduino object and configure digital pin 12 to connect to a push button with mode `pullup`.

```
a = arduino;
configureDigitalPin(a,12,'pullup');
```

Read the state of the button.

```
buttonState = readDigitalPin(a,12)
```

```
buttonState =

     1
```

Change the state by pressing on the push button and read it again.

```
buttonState = readDigitalPin(a,12)
```

```
buttonState =

     0
```

Play a Tone on a Piezo Speaker

This example shows how to configure a piezo speaker on digital pin and play a tone at a specified frequency for a specified amount of time.

Connect a piezo speaker to pin 11 on your Arduino.

Create an Arduino object.

```
a = arduino;
```

Play a tone at 2400Hz frequency for 10 seconds.

```
playTone(a,11,2400,10);
```

To stop playing the tone, change the duration to 0 seconds or change the frequency to 0 Hz.

```
playTone(a,11,2400,0);
```

Control LEDs

This example shows how to configure LEDs to turn on and off and to dim and brighten using MATLAB commands.

Connect a an LED to pin 9 on your Arduino.

Create an Arduino object.

```
a = arduino;
```

Write data to the digital pin to turn the LED on and off repeatedly for 10 seconds.

```
for idx = 0:10
    writeDigitalPin(a,9,1);
    pause(0.5);
    writeDigitalPin(a,9,0);
    pause(0.5);
end
```

Change the brightness from maximum to minimum using the digital pins PWM duty cycle.

```
for brightness = 1:-0.1:0
    writePWMDutyCycle(a,9,brightness);
    pause(0.5);
end
```

I2C Sensors

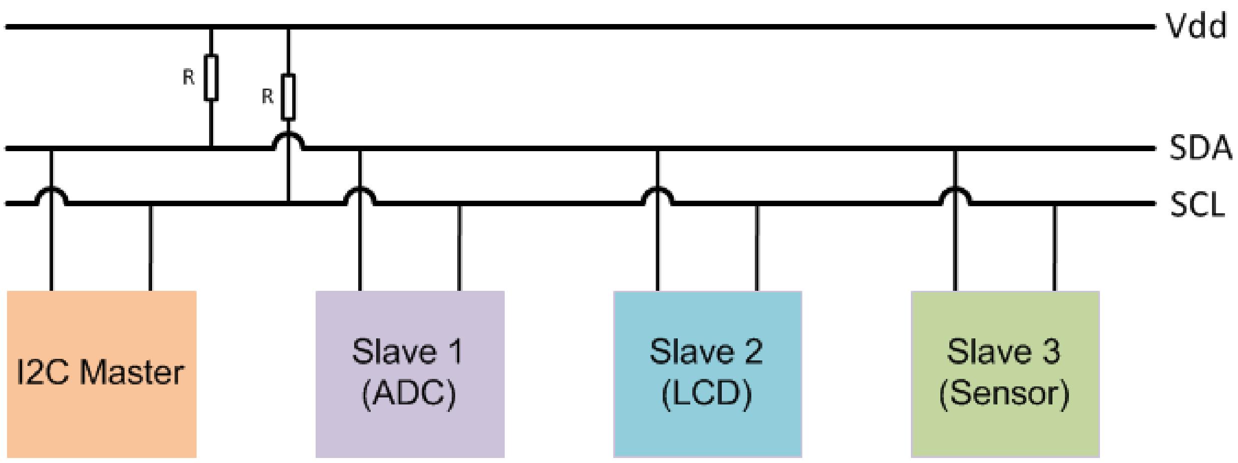
Arduino I2C Interface

I2C, or Inter-Integrated Circuit, is a chip-to-chip protocol for communicating with low-speed peripherals. MATLAB Support Package for Arduino Hardware includes the I2C library which creates an interface to communicate with I2C devices. Each Arduino board has specific pins I2C interface. Refer to your hardware specifications to locate the correct pins.

You can use I2C devices in many applications including:

- real-time clocks
- digital potentiometers
- temperature sensors
- digital compasses
- memory chips
- FM radio circuits
- input/output expanders
- LCD controllers
- amplifiers

Arduino hardware has one or two I2C buses. Each bus has an I2C Master connected to two bidirectional lines, serial data line (SDA), and serial clock (SCL). These two lines are connected to a pair of pins on the hardware. You can connect multiple I2C devices, such ADCs, LCDs, and sensors, to the I2C pins on the Arduino hardware. Each I2C device on an I2C bus must have a unique address. Most devices have a default address assigned by the manufacturer. If the address is not unique, refer to the device data sheet and reconfigure the address. Often, you can reconfigure the address using a pair of jumpers on the device. MATLAB Support Package for Arduino Hardware only supports 7-bit addressing.



I2C Functions

`arduino` `i2cdev` `scanI2CBus` `readRegister` `write` `writeRegister` `arduinoExamples`

Communicate With an I2C EEPROM Device

This example shows how to store and retrieve data from an EEPROM on an I2C device. Using EEPROM you can read, erase, and rewrite individual bits of data from the sensor's memory. Before using this example, wire an EEPROM to the I2C pins on your Arduino.

Create an I2C connection on your Arduino.

Create an Arduino object and include the I2C library.

```
arduino('com22', 'uno', 'Libraries', 'I2C');
```

Scan for I2C address on the Arduino.

```
addrs = scanI2CBus(a)
```

```
addrs =
'0x48'
```

Create an I2C device object using the address and the Arduino object.

```
td = i2cdev(a, '0x48');
```

The bus number defaults to 0. If you have hardware that has dedicated I2C interfaces SDA1 and SCL1, you may need to specify bus number 1.

Write “Hello World!” to the device and read it back.

Write “Hello World!” to the device register at address 0.

```
    write(eeprom,[0 'Hello World!']);
```

Write “Hello World!” to the device register at address 8.

```
   write(eeprom,[8 'Hello World!']);
```

Enable EEPROM and read 12 bytes back.

```
    write(eeprom,0);
    char(read(eeprom,12))'

    ans =
    Hello World!
```

SPI Sensors

Arduino and SPI Interface

SPI, or Serial Peripheral Interface is a full-duplex serial protocol for communicating with high-speed peripherals, such as microcontrollers. You can communicate SPI devices and sensors via SPI interface on Arduino boards using MATLAB Support Package for Arduino Hardware. You can also use this interface to communicate between two microcontrollers. Typically, SPI devices have one master device controlling all peripheral devices. The SPI library has three common pins that are hardwired and a dedicated pin for output.

- MISO, receives data from the SPI peripheral device
- MOSI, which outputs data to the SPI peripheral device
- SCLK, which outputs a serial clock signal to synchronize communications
- SS, which enables and disables peripheral devices from the master

Every SPI device implements SPI standards differently. Refer to the device datasheet to understand how your device will work. SPI Devices transmit data in four basic modes and control the clock phase and the clock polarity. For more information refer to the SPI standards documentation.

SPI Functions

```
spidev writeRead arduinoExamples
```

Servo Motors

Servo Motors

Servo motors have integrated circuitry inside the motor unit. The shaft is typically is fitted with a gear and can be positioned as needed. You can use MATLAB Support Package for Arduino Hardware to control movement of the shaft.

You control servos using PWM signals. The motor turns 90° in either direction based on the maximum and minimum pulse width and the pulse-repetition rate.

There are two types of servos:

- **AC**: Heavy duty motors typically used in industrial machinery. These motors can handle high level of current surges.
- **DC Motor**: Built for smaller applications and continuous rotation. The output shaft houses two ball bearings which reduce friction and give you easy access to the rest-point adjustment potentiometer.

You can use the servo library on MATLAB to work with small and hobby DC motors.

Servo Functions

`arduino` `servo` `readPosition` `writePostion` `arduinoExamples`

Rotate a Servo Motor

This example shows how to rotate a servo motor by its maximum angle.

Create an Arduino object and include the servo library.

```
a = arduino('com22', 'uno', 'Libraries', 'Servo');
```

Configure a servo object using the PWM pin 5 and set the minimum pulse duration to `1e-3` seconds and the maximum pulse durations to `2e-3` seconds.

```
s = servo(a,5, 'MinPulseDuration', 1e-3, 'MaxPulseDuration',2e-3);
```

Write 1, to turn the motor by the maximum angle.

```
writePosition(s, 1);
```

The angle depends on device pulse duration. Refer to your device data sheet for valid pulse duration values.

Add-On Shields

Add-On Shields

Add-on shields are boards you can plug on to Arduino hardware and extend the hardware capabilities. With Shields you can control motors from your board. Currently you can use Adafruit Motorshields V2 with MATLAB Support Package for Arduino Hardware.

Add-On Functions

`arduino` `addon` `dcmotor` `servo` `start` `stop` `stepper` `move` `release` `arduinoExamples`

Functions — Alphabetical List

addon

Create add-on library connection

Syntax

```
dev = addon(a,lib)
dev = addon(a,lib,Name,Value)
```

Description

`dev = addon(a,lib)` creates a connection to the specified add-on library on the Arduino hardware `a`.

`dev = addon(a,lib,Name,Value)` creates an add-on library connection with additional options specified by one or more Name,Value pair arguments.

Examples

Create an Add-On Library Object

Create an Arduino object and create the Adafruit Motor Sheild V2 object.

```
a = arduino()
dev = addon(a,'Adafruit\MotorshieldV2')
```

```
dev =

motorshieldv2 with properties:

            Pins: A4(SDA), A5(SCL)
      I2CAddress: 96 (0x60)
    PWMFrequency: 1600 (Hz)
```

Specify I2C Address and PWM Frequency to an Add-On Library

Create an arduino object and create the Adafruit Motor Sheild V2 object with an I2C address of `0x61` and PWM frequency of `1200`.

```
a = arduino()
dev = addon(a,'Adafruit\MotorshieldV2','i2caddress', '0x61' 'pwmfrequency', 1200)

dev =

motorshieldv2 with properties:

            Pins: A4(SDA), A5(SCL)
      I2CAddress: 97 (0x61)
    PWMFrequency: 1200 (Hz)
```

Input Arguments

a — Arduino hardware connection
object

Arduino hardware connection created using `arduino`, specified as a object.

`lib` — Add-on library name
string

Add-on library vendor and name specified as a string. You must specify both the add-on library vendor and the name. Currently only Motor Shield V2 from Adafruit is supported, which must be specified as `'Adafruit\MotorShieldV2'`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `dev = addon(a, ‘Adafruit\MotorshieldV2’,‘i2caddress’, ‘0x61’ ‘pwmfrequency’, 3200);`

`'i2caddress'` — I2C device address
numeric | string

I2C bus address specified as a number a string. You can specify either a hexadecimal value or a scalar integer. The default value is 90 (0x60).

Example: `shield = addon(a, ‘Adafruit \MotorshieldV2’,‘i2caddress’,0x61);` sets the I2C address to `97` or `0x61`.

`'pwmfrequency'` — Add-on device PWM frequency
numeric

Add-on device PWM frequency specified as a number.

Example: `shield = addon(a, ‘Adafruit\MotorshieldV2’,‘pwmfrequency’, 3200);` sets the PWM frequency to 3200.

Output Arguments

`dev` — Add-on library connection
object

Add-on library connection, returned as an object.

See Also

`arduino` | `dcmotor` | `servo` | `stepper`

arduino

Connect to Arduino hardware

Syntax

```
a = arduino
a = arduino(port,board)
a = arduino(port,board,Name,Value)
```

Description

`a = arduino` creates a connection to the first official Arduino hardware installed on your system.

`a = arduino(port,board)` creates a connection to unoffical (clone) Arduino hardware on the specified port.

`a = arduino(port,board,Name,Value)` creates a connection with additional options specified by one or more Name,Value pair arguments.

Examples

Connect to Arduino hardware

Connect to an Arduino Uno on port 3.

```
a = arduino('com3','uno')
```

```
a = 

  arduino with properties:

                    Port: 'COM3'
                   Board: 'Uno'
     AvailableAnalogPins: [0 1 2 3 4 5]
    AvailableDigitalPins: [2 3 4 5 6 7 8 9 10 11 12 13]
```

```
              Libraries: {'I2C'  'SPI'  'Servo'}
```

Connect to an Arduino on a Mac

Connect to an Arduino Uno on port `/dev/tty.usbmodem1421`.

```
a = arduino('/dev/tty.usbmodem1421','uno')
```

```
a = 

  arduino with properties:

                   Port: '/dev/tty.usbmodem1421'
                  Board: 'Uno'
     AvailableAnalogPins: [0 1 2 3 4 5]
    AvailableDigitalPins: [2 3 4 5 6 7 8 9 10 11 12 13]
              Libraries: {'I2C'  'SPI'  'Servo'}
```

Specify an Arduino Library

Limit the Arduino to use only an I2C library.

```
a = arduino('com3','Uno','libraries','I2C')
```

```
a = 

  arduino with properties:

         Port: 'COM3'
        Board: 'Uno'
    Libraries: {'I2C'}
```

Input Arguments

`port` — Hardware port
string

Hardware port specified as a string.

Example: `a = arduino(‘com5’)`

`board` — Name of Arduino enabled board
string

Name of the Arduino enabled board specified as a string.

Example:

`a = arduino(‘com5’,‘Uno’)` creates a connection to an Arduino Uno board using port 5.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `a = arduino('com3','Uno','libraries','I2C')`

`'libraries'` — Name of Arduino library

`{'I2C' 'Servo' 'SPI'}` (default) | string

Name of the Arduino library specified as the comma-separated pair consisting of 'libraries' and a string. Libraries installed by default extend the Arduino environment.

Example: `a = arduino('com9','libraries','spi')` limits the environment to the specified library.

Example: `a = arduino('com9','libraries','Adafruit/MotorShieldV2')` only includes the Adafruit Motor Shield V2 library and its dependent libraries

.

Output Arguments

a — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, returned as an object.

See Also

`listArduinoLibraries`

arduinoExamples

Open featured examples

Syntax

```
arduinoExamples
```

Description

`arduinoExamples` opens featured examples for MATLAB Support Package for Arduino Hardware.

See Also

`arduino`

configureAnalogPin

Set analog pin mode

Syntax

`configureAnalogPin(a,pin,mode)`

Description

`configureAnalogPin(a,pin,mode)` sets the specified analog `pin` on the Arduino hardware in connection `a` to the specified `mode.` To get the pin mode, omit the mode argument from the input arguments.

Examples

Configure analog pin to input mode

```
a = arduino();
configureAnalogPin(a,2,'input');
```

Input Arguments

`a` — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Analog pin number

numeric

Analog pin number on the physical hardware, specified as a number.

`mode` — Analog pin mode

`'unset'` (default) | string

Analog pin mode specified as a string. Valid pin modes are:

- `input`
- `i2c`

See Also

`arduino` | `readVoltage`

configureDigitalPin

Set digital pin mode

Syntax

```
configureDigitalPin(a,pin,mode)
```

Description

`configureDigitalPin(a,pin,mode)` sets the specified digital `pin` on the Arduino hardware in connection `a` to the specified `mode`. To get the pin mode, omit the mode argument from the input arguments.

Examples

Configure Digital Pin to Output Mode

```
a = arduino();
configureDigitalPin(a,3,'ouput');
```

Input Arguments

a — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Digital pin number

numeric

Digital pin number on the physical hardware, specified as a number.

`mode` — Digital pin mode

`'unset'` (default) | string

Digital pin mode specified as a string. Valid pin modes are:

- `input`
- `output`
- `pullup`
- `pwm`
- `servo`
- I2C
- SPI

Pins are configured on first usage. You can rest the pin mode to change the mode. If you want to use a pull up, you must set the mode to `'pullup'`.

See Also

`arduino` | `readDigitalPin` | `writeDigitalPin`

dcmotor

Attach DC motor to Adafruit motor shield

Syntax

```
dcm = dcmotor(shield,motornum)
dcm = dcmotor(shield,motornum,Name,Value)
```

Description

`dcm = dcmotor(shield,motornum)` attaches a DC motor to the specified port on an Adafruit motor shield connected to an Arduino.

`dcm = dcmotor(shield,motornum,Name,Value)` attaches a DC motor with additional options specified by one or more Name,Value pair arguments.

Examples

Create a DC Motor Connection on an Adafruit

Add a DC motor to an Adafruit motor shield connected to an Arduino hardware.

Connect to an Arduino hardware and create an add-on connection to an Adafruit shield.

```
a = arduino('COM7', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2');
shield = addon(a, 'Adafruit/MotorShieldV2');
```

Create a DC motor connection to port 1 on the shield.

```
dcm = dcmotor(dev, 1)
```

```
dcm = 

  dcmotorv2 with properties:

    MotorNumber: 1 (M1)
           Pins: 'A4, A5'
```

```
          Speed: 0.00
      IsRunning: 0
```

Create a DC Motor Connection on an Adafruit and Specify Speed

Add a DC motor to an Adafruit motor shield connected to Arduino hardware.

Connect to Arduino hardware and create an add-on connection to an Adafruit shield.

```
a = arduino('COM7', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2');
shield = addon(a, 'Adafruit/MotorShieldV2');
```

Create a DC motor connection to port `1` on the shield and set the speed to `0.2`.

```
dcm = dcmotor(dev,1,'Speed',0.2)
```

```
dcm = 

  dcmotorv2 with properties:

    MotorNumber: 1 (M1)
          Speed: 0.20
      IsRunning: 0
```

Input Arguments

`shield` — Add-on library connection
object

Add-on library connection created using `addon`, specified as an object.

`motornum` — Port number on shield
numeric

Motor number on the shield to connect the motor to, specified as a number. Valid values are `1`, `2`, `3`, and `4`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `dcm = dcmotor(shield,1,'Speed'0.2)`

`'speed'` — DC motor speed
0 (default) | numeric

DC motor speed specified as a number between `–1` and `1`.

Example: `dcm = dcmotor(shield,1,'Speed'0.2)` sets the speed of the DC motor attached to port 1 on the motor shield to `0.2`.

Output Arguments

`dcm` — DC motor object
object

DC motor connection returned as an object.

See Also

`addon` | `arduino` | `start` | `stop`

i2cdev

Device connected on Arduino I2C bus

Syntax

```
dev = i2cdev(a,address,Name,Value)
```

Description

`dev = i2cdev(a,address,Name,Value)` creates an object that represents the connection to the device connected to the I2C bus on the specified Arduino hardware `a`.

Examples

Create I2C device Connection

Create Arduino hardware object and scan for the I2C bus.

```
a = arduino('com9');
scanI2CBus(a)
```

```
ans =
   '0x61'
```

Use the address supplied to create an I2C device connection.

```
dev = i2cdev(a,'0x61'))
```

```
dev =
  i2cdev with properties:

        Bus: 0
    Address: '0x61'
       Pins: 'A4(SDA), A5(SCL)'
```

Create I2C device Connection with Additional Options

Create a connection and specify the bus number.

```
dev = i2cdev(a, '0x61','bus',1)
```

```
dev =
  i2cdev with properties:

        Bus: 1
    Address: '0x61'
       Pins: 'A4(SDA), A5(SCL)'
```

Input Arguments

a — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`address` — address of device connected to I2C bus

string

Address of device connected to the Arduino I2C bus, specified as a string. The address string can be a hex or a scalar integer.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `dev = i2cdev(a,'0x71','bus',0)`

`'bus'` — I2C device bus

0 (default) | numeric

I2C device bus specified as the comma-separated pair consisting of "bus" and a number.

If you are using an Arduino Due, use bus 0 to connect to SDA and SDI and bus 1 to connect to SDa1 and SDI1.

Example: `i2cdev = (a,'0x61','bus',1)` creates a connection to the I2C bus `1`.

Output Arguments

`dev` — I2C device connection
object

I2C device connection, returned as an object.

See Also

Functions
`arduino` | `scanI2Cbus` | `spidev`

listArduinoLibraries

Display a list of installed Arduino libraries

Syntax

```
lib = listArduinoLibraries();
```

Description

`lib = listArduinoLibraries();` creates a list of available Arduino libraries and saves the list to the variable `lib`.

Examples

List Arduino libraries

List libraries installed on your system.

```
lib = listArduinoLibraries()
```

```
lib = 

    'Adafruit/MotorShieldV2'
    'I2C'
    'SPI'
    'Servo'
```

Output Arguments

`lib` — List of Arduino libraries
object

List of available Arduino libraries specified as an object or a cell array of strings.

See Also

`arduino`

move

Move stepper motor

Syntax

```
move(sm,steps)
```

Description

`move(sm,steps)` moves the stepper motor for the specified number of steps.

The `move` command blocks MATLAB while the stepper is being moved.

Examples

Move the stepper motor

Connect to Arduino hardware and create an add-on connection to an Adafruit shield.

```
a = arduino('COM7', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2');
shield = addon(a, 'Adafruit/MotorShieldV2');
```

Create a stepper object at port 1 with 200 steps per revolution and 10 RPMs.

```
sm = stepper(shield,1,200,'RPM',10);
```

Move the motor by 10 steps.

```
move(sm, 10);
```

The command now blocks MATLAB while the move executes.

Input Arguments

`sm` — Stepper motor object

object

Stepper motor connection specified as an object, created using `stepper`.

`steps` — Number of steps
number

Number of steps specified as a number that the stepper motor should move.

See Also

`addon` | `arduino` | `release` | `stepper`

playTone

Play tone on piezo speaker

Syntax

```
playTone(a,pin,frequency,duration)
```

Description

`playTone(a,pin,frequency,duration)` plays a tone on a piezo speaker attached to the Arduino hardware at the specified pin and frequency for the specified duration.

Examples

Play a Tone on a Piezo Speaker

Play a tone connected to pin 5 on the Arduino for 30 seconds at 2400Hz.

```
a = arduino()
playTone(a,5,2400,30)
```

Input Arguments

a — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Digital pin number

numeric

Digital pin number on the physical hardware, specified as a number.

`frequency` — Frequency of tone

numeric

Frequency of tone to be played, specified as a number between the frequency range `0` and `32767`Hz.

`duration` — Duration of tone
numeric

Duration of tone to be played specified as a number, which represents the number of seconds. Valid values are between `0` and `30`.

See Also
`arduino`

readDigitalPin

Read data from digital pin on Arduino hardware

Syntax

```
value = readDigitalPin(a,pin)
```

Description

`value = readDigitalPin(a,pin)` reads data from the specified `pin` on the Arduino hardware in connection `a`.

Examples

Read the Value of a Digital Pin

Create a connection to Arduino hardware and read digital pin 13.

```
a = arduino()
readDigitalPin(a,13);
```

Input Arguments

a — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Digital pin number

numeric

Digital pin number on the physical hardware, specified as a number.

Output Arguments

`value` — value acquired from digital pin
logical

Value acquired from digital pin, returned as a logical value.

See Also

`arduino` | `configureDigitalPin` | `writeDigitalPin`

readVoltage

Read Arduino analog pin voltage

Syntax

```
voltage = readVoltage(a,pin)
```

Description

`voltage = readVoltage(a,pin)` reads the voltage on the specified analog input pins on Arduino hardware.

Examples

Read Voltage from Analog Pin

Create an arduino object and read voltage from and analog input pin.

```
a = arduino;
readVoltage(a, 4)
```

```
ans =

    2.1533
```

Input Arguments

a — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Analog pin number

numeric

Analog pin number on the physical hardware, specified as a number.

Output Arguments

`voltage` — Voltage read from analog pin

numeric

Voltage read from an analog pin on an Arduino hardware specified as a numeric double.

See Also

`arduino` | `configureAnalogPin`

readPosition

Read servo motor position

Syntax

```
position = readPosition(s)
```

Description

`position = readPosition(s)` reads the shaft position as a ratio

of 1 to the maximum angle of the specified servo motor...

Examples

Read servo motor position

Create servo object.

```
a = arduino();
s = servo (a, 3);
```

Read the position of the servo.

```
postion = readPosition(s)
```

```
position =

    0.5
```

Input Arguments

s — Servo object
object

Servo object connected to Arduino hardware specified as an object.

Output Arguments

`position` — Position of servo motor
numeric

Position of the servo motor specified as a number representing the angle from `0` to `1` .

See Also

`arduino` | `servo` | `writePosition`

readRegister

Read from I2C device register

Syntax

```
out = readRegister(dev,register,precision)
```

Description

`out = readRegister(dev,register,precision)` reads from the specified I2C register with option precision parameters.

Examples

Read From I2C Device Register

Create an arduino object and attach an I2C device object to it.

```
a = arduini();
dev = i2cdev(a, '0x61');
```

Read from register at address 20.

```
value = readRegister(dev,'20');
```

Specify Precision To Read from I2C Register

Create an arduino object and attach an I2C device object to it.

```
a = arduino();
dev = i2cdev(a, '0x61');
```

Read from register at address 20 with precision of uint16.

```
value = readRegister(dev,'20','uint16');
```

Input Arguments

`dev` — I2C device connection

object

I2C device connection represented as an object.

`register` — Address of register on I2C device

double | string

Address of the register on the I2C device specified as a double or a string. The value must match the address specified in the I2C device data sheet.

`precision` — Data precision

| string

Data precision, specified as a string. Match the data precision to the size of the register on the device. Data precision defaults to `uint8` and you can set it to:

Data Types: `uint8` | `uint16`

Output Arguments

`out` — Value of data

numeric

Value of data stored at register, returned as a numeric array.

See Also

`arduino` | `i2cdev` | `writeRegister`

release

Release stepper motor

Syntax

```
release(sm)
```

Description

`release(sm)` release the stepper motor allowing it to move freely.

Examples

Release the stepper motor

Connect to Arduino hardware and create an add-on connection to an Adafruit shield.

```
a = arduino('COM7', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2');
shield = addon(a, 'Adafruit/MotorShieldV2');
```

Create a stepper object at port 1 with 200 steps per revolution and 10 RPMs.

```
sm = stepper(shield,1,200,'RPM',10);
```

Move the motor by 10 steps and release it.

```
move(sm, 10);
release(sm);
```

Input Arguments

`sm` — Stepper motor object

object

Stepper motor connection specified as an object, created using `stepper`.

See Also

`addon` | `arduino` | `stepper`

scanI2CBus

Scan Arduino device for I2C bus address

Syntax

`addr = scanI2CBus(a)`

`addr = scanI2CBus(a, Name, Value)`

Description

`addr = scanI2CBus(a)` scans the Arduino device in object `a` and stores it in the variable `addr`.

`addr = scanI2CBus(a, Name, Value)` scans a specific bus. For Name, enter 'bus'. For Value, enter 0 or 1. If using SDA1 and SCL1 pins on Due board, set it to 1.

Examples

Scan for I2C bus on an Arduino device

```
a = arduino('com9');
scanI2CBus(a)
```

```
ans =
    '0x48'
```

Input Arguments

a — Arduino device connection
object Arduino device connection created using `arduino`, specified as an object.

Output Arguments

`addr` — I2C bus address
string

I2C bus address represented as a string or a cell array of strings.

servo

Create connection servo motor

Syntax

```
s = servo(a, pin)
s = servo(a, pin,Name,Value)
```

Description

`s = servo(a, pin)` creates a servo motor object connected to the specified pin on the Arduino hardware `a`.

`s = servo(a, pin,Name,Value)` creates a servo motor object with additional options specified by one or more Name,Value pair arguments.

Examples

Create a Servo Object

Create a servo object using pin 3.

Create an arduino object and attach the servo object to pin 3.

```
a = arduino()
s = servo(a,3)
```

```
s = 

  Servo with properties:

               Pins: 4
   MinPulseDuration: 5.44e-04 (s)
   MaxPulseDuration: 2.40e-03 (s)
```

Specify Minimum and Maximum Pulse Duration for a Servo

Set the minimum duration to 7e-4 and maximum to 2.3e-3 seconds.

Create an arduino object and attach the servo object to pin 3 and specify

```
a = arduino()
s = servo(a,3,'MinPulseDuration', 7.00e-4,'MaxPulseDuration',2.3e-3)
```

```
s =

  Servo with properties:

                 Pins: 3
    MinPulseDuration: 7.00e-04 (s)
    MaxPulseDuration: 2.30e-03 (s)
```

Input Arguments

a — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Digital pin number

numeric

Digital pin number on the Arduino board, specified as a number.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `s= servo(a,7,‘MaxPulseDuration’, 2e-3, ‘MinPulseDuration’, 1e-3);`

`'MinPulseDuration'` — Pulse duration of servo

5.44e-4 seconds (default) | numeric

The minimum pulse duration specified as a numeric number in seconds and is equal to the width of the pulse required to put the motor at its minimum angle. Refer to the device datasheet for correct values.

`'MaxPulseDuration'` — Pulse duration of servo

2.4e-3 seconds (default) | numeric

The maximum pulse duration specified as a numeric number in seconds and is equal to the width of the pulse required to put the motor at its maximum angle. Refer to the device datasheet for correct values.

Output Arguments

`s` — Servo object

object

Servo object returned as an object.

See Also

`arduino` | `readPosition` | `writePosition`

servo

Create add-on servo motor connection

Syntax

```
s = servo(shield, motornum)
```

Description

`s = servo(shield, motornum)` creates a servo motor object on the specified add-on shield motor number.

Examples

Create a Servo Motor Object Using an Add-On Library

Create an arduino object and load the Adafruit library.

```
a = arduino('com9','Uno','libraries','Adafruit\MotorshieldV2');
```

Create the add-on library object.

```
shield = addon(a, ‘Adafruit\MotorshieldV2’);
```

Attach a servo to the add-on shield.

```
s = servo(shield,1);
```

Input Arguments

`shield` — Add-on library connection

object

Add-on library connection created using `addon`, specified as an object.

`motornum` — Servo motor number
numeric

Servo motor number specified as a number. Valid values are 1 and 2. Motor number 1 uses digital pin 10 on the Arduino board. Motor number 2 uses digital pin 9.

Output Arguments

`s` — Servo object
object

Servo object returned as an object.

See Also

`addon` | `arduino`

spidev

Connect to SPI device

Syntax

```
dev = spidev(a,cspin)
dev = spidev(a,cspin,Name,Value)
```

Description

`dev = spidev(a,cspin)` creates a connection to the SPI device on the specified Arduino board and chip select pin.

`dev = spidev(a,cspin,Name,Value)` creates a connection to the SPI device with additional options specified by one or more Name,Value pair arguments.

Examples

Create SPI Connection to an Arduino Board at CS Pin 4

Create an Arduino connection and connect it to an SPI device

```
a = arduino('com9');
dev = spidev(a, 4);
```

Create an SPI Connection with Communication Mode and Bit Order Parameters

Set the communication mode to `3` and the bit order to `lsbfirst`.

```
a = arduino('com9');
dev = spidev(a, 4, 'spimode', 3, 'bitorder', 'lsbfirst');
```

Input Arguments

a — Arduino hardware connection
object

Arduino hardware connection created using `arduino`, specified as an object.

`cspin` — Chip select pin number
numeric

Chip select pin number to used to communicate with the SPI device.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `dev = spidev(a, 4, 'spimode', 3, 'bitorder', 'lsbfirst','mode',3);`

`'bitorder'` — SPI communication bit order
msbfirst (default) | string

SPI communication bit order specified as a string. Possible values are:

- `'msbfirst'`, where the most significant bit is in the first column.
- `'lsbfirst`, where the least significant bit is in the first column.

`'mode'` — SPI communication mode
numeric

SPI communication mode specifying clock polarity and phase, specified as the comma-separated pair consisting of `'mode'` and an number with a value between `0` and `3`.

Output Arguments

`dev` — SPI device connection
object

SPI device connection returned as an object.

See Also

`arduino` | `i2cdev`

start

Start DC motor

Syntax

```
start(dcm)
```

Description

`start(dcm)` starts the configured DC motor.

Examples

Start a DC motor

Add a DC motor to an Adafruit motor shield connected to Arduino hardware and start the motor.

Connect to Arduino hardware and create an add-on connection to an Adafruit shield.

```
a = arduino('COM7', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2');
shield = addon(a, 'Adafruit/MotorShieldV2');
```

Create a DC motor connection to port 1 on the shield.

```
dcm = dcmotor(shield, 1)
```

```
dcm = 

  dcmotorv2 with properties:

    MotorNumber: 1 (M1)
          Speed: 0.00
      IsRunning: 0
```

Change the speed to 0.2, start the DC motor and display the motor object.

```
dcm.Speed = 0.2
start(dcm)
dcm

dcm = 

  dcmotorv2 with properties:

    MotorNumber: 1 (M1)
          Speed: 0.20
      IsRunning: 1
```

Input Arguments

`dcm` — DC motor object

object

DC motor connection specified as an object, created using `dcmotor`.

See Also

`dcmotor` | `stop`

stepper

Attach stepper motor to Adafruit motor shield

Syntax

```
sm = stepper(shield,motornum,sprev)
sm = stepper(shield,motornum,sprev,Name,Value)
```

Description

`sm = stepper(shield,motornum,sprev)` creates a stepper motor object connected to the specified port on the Adafruit motor shield on the specified motor num on the shield with specified steps per revolution.

`sm = stepper(shield,motornum,sprev,Name,Value)` creates a stepper motor object with additional options specified by one or more Name-Value pair arguments.

Examples

Create a stepper object

Specify a port and steps per revolution.

Connect to Arduino hardware and create an add-on connection to an Adafruit shield.

```
a = arduino('COM7', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2');
shield = addon(a, 'Adafruit/MotorShieldV2');
```

Create a stepper motor connection to a motor number 1 on the shield, with an RPM of 10.

```
sm = stepper(dev,1,200,'RPM',10)
```

```
sm =

  stepper with properties:

           MotorNumber: 1
```

```
                 Pins: 'A4, A5'
   StepsPerRevolution: 200
                  RPM: 10
             StepType: Single ('Single', 'Double', 'Interleave', 'Microstep')
```

Create a stepper object and define RPM and step type

Specify RPM and step type option name value pairs.

Connect to Arduino hardware and create an add-on connection to an Adafruit shield.

```
a = arduino('COM7', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2');
shield = addon(a, 'Adafruit/MotorShieldV2');
```

Create a stepper motor connection to port 1 on the shield, with an RPM of 10..

```
sm = stepper(shield,1,200,'RPM',10,'stepType','interleave')
```

```
sm =

  stepper with properties:

          MotorNumber: 1
   StepsPerRevolution: 200
                  RPM: 10
             StepType: Interleave ('Single', 'Double', 'Interleave', 'Microstep')
```

Input Arguments

`shield` — Add-on library connection
object

Add-on library connection created using `addon`, specified as an object.

ÒÛċÛÿ×ĚÒ — J I q I o number on shield
numeric

Motor number on the shield to connect the motor to, specified as a number. Valid values are 1 and 2. If using M1 and M2 on the shield, set motor number to 1.

If using M3 and M4, set motor number to 2

`sprev` — Steps per revolution
numeric

Steps per revolutions of the stepper motor, specified as a number.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `sm = stepper(shield,1,200,'RPM',10,'stepType','interleave')`

`'RPM'` — Revolutions per minute
0 (default) | numeric

Revolutions per minute specified as a number. RPM determines the speed of the motor.

Example: `sm = stepper(shield,1,'RPM',10)` creates a stepper with an RPM of 10.

`'StepType'` — Coil activation type
single (default) | string

Coil activation type specified as a string. Valid types include:

- Single
- Double
- Interleave
- Microstep

Example: `sm = stepper(shield,1,200,'stepType','interleave')` creates a stepper with an interleaved step type.

Output Arguments

`sm` — Stepper motor object
object

Stepper motor connection returned as an object.

See Also

`addon` | `arduino` | `move` | `release`

stop

Stop DC motor

Syntax

```
stop(dcm)
```

Description

`stop(dcm)` stops the specified DC motor.

Examples

Stop a DC motor

Add a DC motor to an Adafruit motor shield connected to Arduino hardware. Start and stop the motor.

Connect to Arduino hardware and create an add-on connection to an Adafruit shield.

```
a = arduino('COM7', 'Uno', 'Libraries', 'Adafruit\MotorShieldV2');
shield = addon(a, 'Adafruit/MotorShieldV2');
```

Create a DC motor connection to port 1 on the shield.

```
dcm = dcmotor(shield, 1);
```

Change the speed to 0.2, start the DC motor and display the motor object.

```
dcm.Speed = 0.2
start(dcm)
dcm
```

```
dcm = 

  dcmotorv2 with properties:
    MotorNumber: 1 (M1)
          Speed: 0.20
      IsRunning: 1
```

Stop the motor and display the object.

```
stop(dcmotor);
dcm
```

```
dcm =

  dcmotorv2 with properties:

    MotorNumber: 1 (M1)
          Speed: 0.20
      IsRunning: 0
```

Input Arguments

`dcm` — DC motor object

object

DC motor connection specified as an object, created using `dcmotor`.

See Also

`arduino` | `dcmotor` | `start`

writePosition

Write position of servo motor

Syntax

```
writePosition(s,position)
```

Description

`writePosition(s,position)` writes the specified `value` to the specified servo on the Arduino hardware.

Examples

Set Servo Motor Position

.

Create servo object.

```
a = arduino();
s = servo (a, 3);
```

Rotate the motor to its mid-position.

```
writePosition(s, 0.5)
```

Input Arguments

`s` — Servo object

object

Servo object connected to an Arduino device specified as an object.

`position` — Position of shaft

numeric

Position of servo motor shaft specified as a number representing the angle from `0` to `1`.

See Also

`arduino` | `readPosition` | `servo`

writePWMDutyCycle

Set digital pin PWM duty cycle

Syntax

```
writePWMDutyCycle(a, pin, dutyCycle)
```

Description

`writePWMDutyCycle(a, pin, dutyCycle)` sets the PWM duty cycleon a digital pin specified for the Arduino hardware `a`.

Examples

Specify the DutyCycle of Digital Pin

Specify a `0.33` duty cycle for an LED attached to digital pin 7 on Arduino hardware.

```
a = arduino();
writePWMDutyCycle(a,7,0.33);
```

Input Arguments

a — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Digital pin number

numeric

Digital pin number on the physical hardware, specified as a number.

`dutyCycle` — Value of the digital pin duty cycle

numeric

Value of digital pin’s PWM duty cycle specified as number between `0` and `1`.

See Also

`arduino` | `writePWMVoltage`

writePWMVoltage

Write digital pin PWM voltage value

Syntax

```
writePWMVoltage(a, pin, voltage)
```

Description

`writePWMVoltage(a, pin, voltage)` writes the specified voltage value to the PWM pin on the Arduino hardware.

Examples

Specify the Voltage of Digital Pin

Specify digital pin 7 on Arduino hardware to have `3` volts.

```
a = arduino();
writePWMVoltage(a,7,3);
```

Input Arguments

a — Arduino hardware connection
object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Digital pin number
numeric

Digital pin number on the physical hardware, specified as a number.

`voltage` — Voltage of the digital pin
numeric

Voltage of digital pin’s PWM specified as number between `0` and `5` volts. Check your hardware data sheet for accepted voltage ranges. For example, Arduio Uno accepts 0 – 5 `V` and Arduino Due accepts `0 – 3.3 V`.

See Also

`arduino` | `writePWMDutyCycle`

write

Write data to I2C bus

Syntax

```
write(dev,value,precision)
```

Description

`write(dev,value,precision)` writes the specified data to the I2C bus.

Examples

Write data to an I2C device on an Arduino

Create a connection to an I2C device on an Arduino.

```
a = arduino();
dev = i2cdev(a, 7);
```

Write data and read it back

```
write(dev,dataIn)
```

Input Arguments

`dev` — I2C device connection
object

I2C device connection represented as an object.

`value` — Data to write to I2C device
numeric array

Data to write to I2C device specified as a double. Write the data to the device before you can read it.
precision - Data precision to be write to the I2C device Valid values are 'uint8' and 'uint16'.

writeDigitalPin

Write to digital pin on Arduino hardware

Syntax

`writeDigitalPin(a,pin,value)`

Description

`writeDigitalPin(a,pin,value)` writes the specified `value` to the specified `pin` on the Arduino hardware in the connection `a`.

Examples

Write to Digital Pin

Write a value to digital pin 7 with an LED to turn it on.

```
a = arduino();
writeDigitalPin(a, 7, 1);
```

Input Arguments

`a` — Arduino hardware connection

object

Arduino hardware connection created using `arduino`, specified as an object.

`pin` — Digital pin number

numeric

Digital pin number on the hardware, specified as a number.

`value` — Value of the digital data

logical

Value of digital data to write to the specified pin on a hardware, specified as a logical value of `0` and `1` or `true` and `false`.

See Also

`arduino` | `configureDigitalPin` | `readDigitalPin`

writeRead

Read and write data from SPI sensor

Syntax

```
dataOut = writeRead(dev,dataIn,precision)
```

Description

`dataOut = writeRead(dev,dataIn,precision)` reads the specified data, written to the SPI device as `dataIn`.

Examples

Read and Write Data from an SPI Device on an Arduino Board

Create a connection to an SPI device on an Arduino board.

```
a = arduino();
dev = spidev(a, 7);
```

Write data and read data.

```
dataIn = [2 0 0 255];
dataOut = writeRead(dev,dataIn);
```

Input Arguments

`dev` — SPI device connection

object

SPI device connection represented as an object.

`dataIn` — Data to write to SPI device

numeric array

Data to write to SPI device specified as a numeric array. Write the data to the device before you read it.

`precision` — Data precision
string

Data precision, specified as a string. Match the data precision to the size of the register on the device. Data precision defaults to `uint8` and you can set it to:

Data Types: `uint8` | `uint16`

Output Arguments

`dataOut` — Data read from SPI device
numeric

Data read from SPI device returned as a numeric array.

See Also

`arduino` | `spidev`

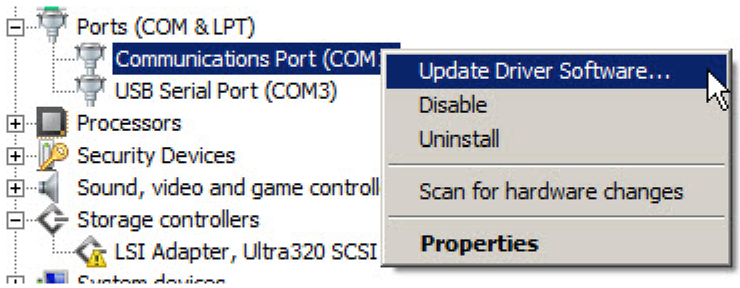
Troubleshooting

Cannot Auto Detect Arduino hardware

If you are using an official Arduino hardware, MATLAB should auto detect the port connected to the hardware on both Windows® and Macintosh systems. If you cannot connect to an official hardware on Windows, you may need to update the hardware driver.

Update Device Driver on a Windows System

1. Open Device Manager and expand the `Ports (COM & LPT)` list.
2. Right click on the port that your device is connected to and select **Update Drive Software**.



3 Select 'Browse my computer for driver software'.

4 Specify the location where the MATLAB Support Package for Arduino Hardware is installed.

Unsupported Device

Check the Supported Device page on the MathWorks® Web site to ensure that the Arduino device you are using is supported.

Manual disconnect

If you have manually disconnected the board and reconnected, you must clear the arduino object from the MATLAB workspace before you reconnect.

Why do I need to specify board type and port?

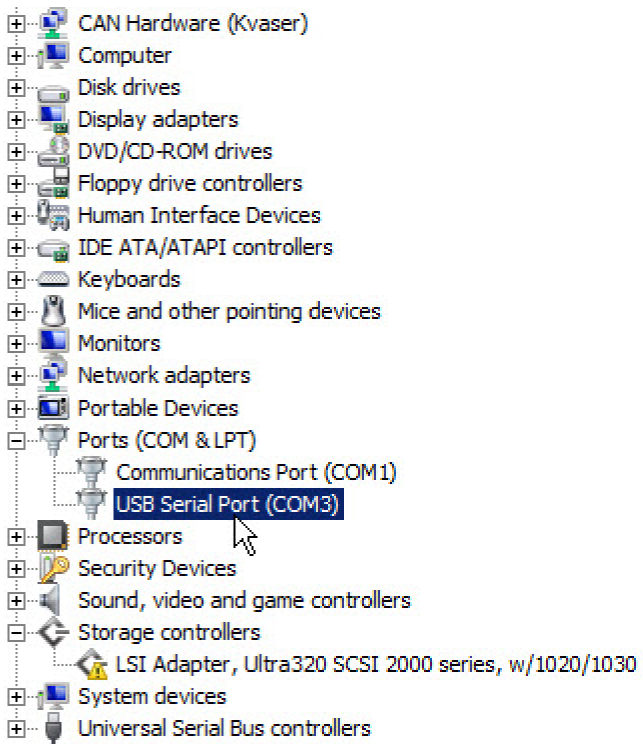
Unofficial Arduino devices, are not automatically recognized and you must specify port and board type to create connection. For a list devices, see the Supported Device page on the MathWorks Web site: http://www.mathworks.com/hardware-support/arduino-matlab.html

After you plug your Arduino hardware, you can find the port number on your system.

Find port number on Windows

1 Open Device Manager and expand the `Ports (COM & LPT)` list.

2 Note the number on the **USB Serial Port**.



CAN Hardware (Kvaser)
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
Floppy drive controllers
Human Interface Devices
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Portable Devices
Ports (COM & LPT)
Communications Port (COM1)
USB Serial Port (COM3)
Processors
Security Devices
Sound, video and game controllers
Storage controllers
LSI Adapter, Ultra320 SCSI 2000 series, w/1020/1030
System devices
Universal Serial Bus controllers

Find port number on Macintosh

1 Open a Mac Terminal and type:

```
'ls /dev/*'
```

2 Note the port number listed for `tty.usbmodem*` or `tty.usbserial*`. The port number is represented with `*` here.

My I2C server code hangs

Check the power line and make sure it is still connected. With I2C devices, the server code can stop working if the power line is disconnected.

I lost my Arduino connection

Unplugged hardware

This is the most common cause of a lost connection. Check your wires and make sure it is still plugged in.

Board with low data memory

If your Arduino has very low data memory, you may lose connection while working large data sets. If this occurs you can delete all libraries you are not using from the board and free up some space.

Too many device objects

If you have configured too many device objects on your Arduino, it may use up memory resources and cause a lost connection. Clear all objects. Then, recreate the arduino object and create fewer peripheral devices.

Pin is Not Receiving Data

Make sure devices that have floating or high impedence have proper pull up or pull down resistors.

I see strange data

Pin not receiving proper signal

Make sure your pins is properly configured. For example, a pull up or a pull down is being used for read or write.

Board with low data memory

If your Arduino has very low data memory, you may see strange data. If this occurs you can delete all libraries you are not using from the board and free up some space.

Incorrect input or output

You can get incorrect readings if you are using the wrong pin. For example if you are writing to a pin configured as input or pullup.

Problems using pulse duration setting on Servo

You can get errors if the pulse duration settings on a Servo are incorrect. Minimum and maximum pulse duration values must be set correctly. Refer to you Servo device specification for appropriate values.

My Servo Pulse is Not Working

Refer to you Servo Motor data sheet for pulse duration values and calibrate accordingly.

Hardware reserved error

You may get a hardware reserved error if there is a conflict between analog input and I2C for pins 4 and 5. The hardware is reserved by whichever operation occurs first. Use configureDigitalPin or configureAnalogPin to reconfigure the pin to a different mode.

Cannot stack motor shields

If you are using an Arduino Due board, you cannot stack multiple motor shields. To be able to stack shields, use Uno or any other supported Arduino hardware.

My Stepper is Not Rotating

Your device may not have enough power. Check the motor's operating voltage and make sure that you have enough power to the device. Then, re-power the device and try again.

My Motor is Not Working

- Check to see if your device has enough power.
- Check to see if the jumper is missing.
- Make sure the device voltage range requirements are met. For example, an Arduino Due board only supports voltage range 0 to 3.3V.
- Refer to the Adafruit FAQ page for more troubleshooting tips at

  https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/faq

I Need More Help

Create a trace file by specifying the TraceOn to true when creating arduino obejct.

Then, rerun your code and inspect commands being executed on Arduino hardware.

Please have the full trace when contacting MathWorks to report an issue.

For example:

```
a = arduino('com3','Uno','TraceOn', true)
```

Examples

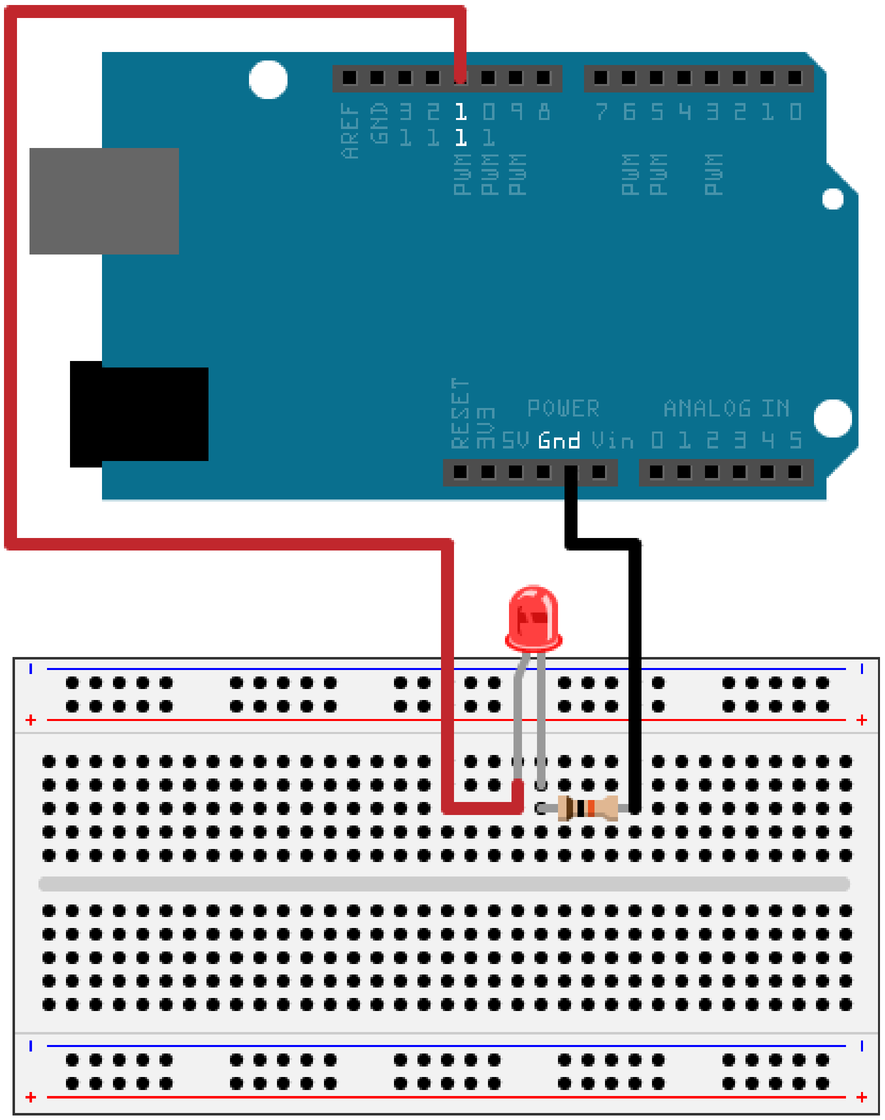
Getting Started with MATLAB Support Package for Arduino Hardware

This example shows how to use MATLAB® Support Package for Arduino® Hardware to perform basic operations on the hardware such as turning an LED on and off, blinking LEDs and playing sound on a speaker.

Contents

Hardware setup

- Connect an LED to digital pin 11 on the Arduino hardware through a 1KOhm resistor.



Create an arduino object

```
a = arduino();
```

If you have more than one Arduino board connected, specify the port and board type.

```
a = arduino('com23', 'uno');
```

Turn LED on and off

Write value `1` or true to digital pin 13 turns on the built-in LED and write a value of 0 or false turns it off. Execute the following command at the MATLAB prompt to turn the LED off and on.

```
writeDigitalPin(a, 13, 0);
pause(2);
writeDigitalPin(a, 13, 1);
```

Configure the LED to blink at a period of 1 second.

```
for i = 1:10
   writeDigitalPin(a, 13, 0);
   pause(0.5);
   writeDigitalPin(a, 13, 1);
   pause(0.5);
end
```

Brighten and dim LED

Send pulse signals of specified width to the PWM pins on the Arduino hardware. PWM signals can light up LEDs connected to the pin. The duty cycle of the pulse controlls the brightness of the LED. Calculate the amount that the LED brightens and dims by dividing the max and min duty cycle for the pin by the number of iterations.

```
brightness_step = (1-0)/20;
for i = 1:20
   writePWMDutyCycle(a, 11, i*brightness_step);
   pause(0.1);
end

for i = 1:20
   writePWMDutyCycle(a, 11, 1-i*brightness_step);
   pause(0.1);
end
```

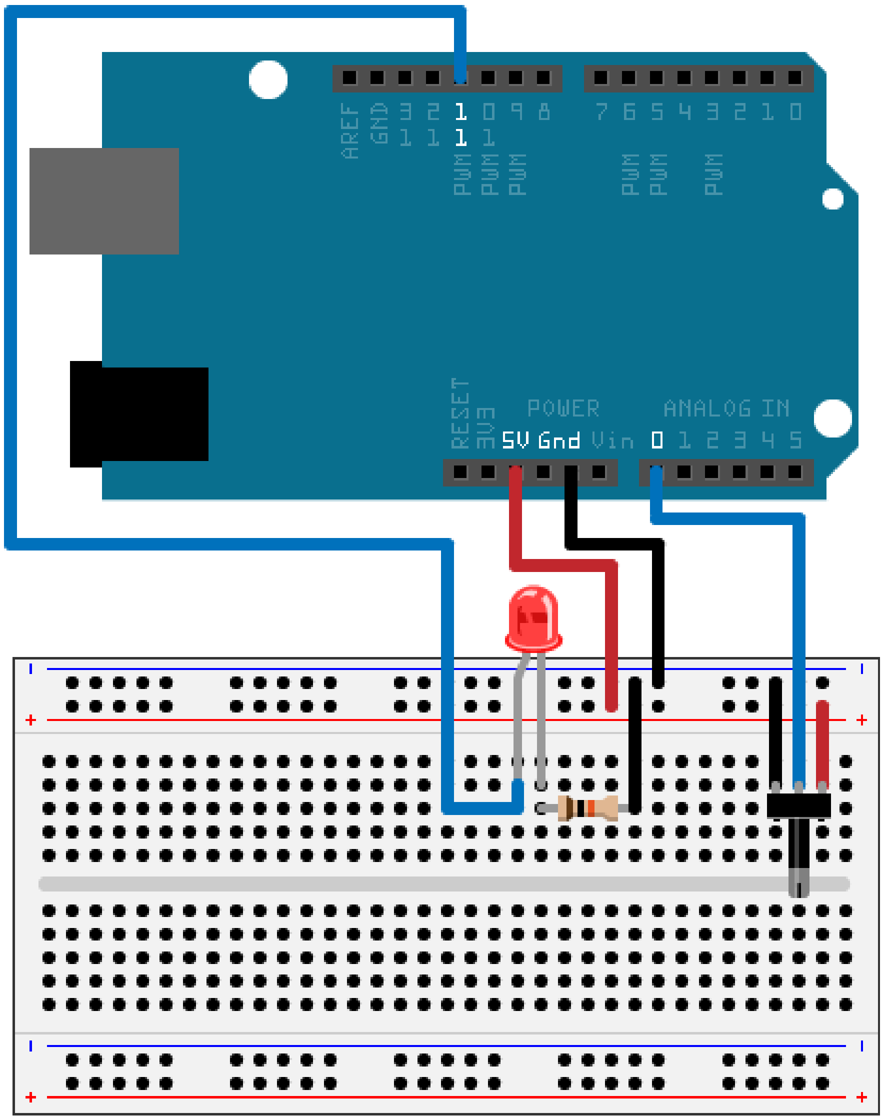
You can also brighten and dim the lights by changing the voltage of the PWM signal. Calculate the amount that the LED brightens and dims by dividing the max and min voltage for the pin by the number of iterations.

```
brightness_step = (5-0)/20;
for i = 1:20
   writePWMVoltage(a, 11, i*brightness_step);
   pause(0.1);
end

for i = 1:20
   writePWMVoltage(a, 11, 5-i*brightness_step);
   pause(0.1);
end
```

Control an LED using a potentiometer

The potentiometer changes the voltage value read from analog pin 0 which can be used to set the voltage level on the PWM pin to control the brightness of the LED connected. Connect a potentiometer to Arduino hardware with the middle leg connected to analog pin 0 and the other two connected to 5V and GND.



```
time = 200;
while time > 0
   voltage = readVoltage(a, 0);
   writePWMVoltage(a, 11, voltage);

   time = time - 1;
   pause(0.1);
end
```

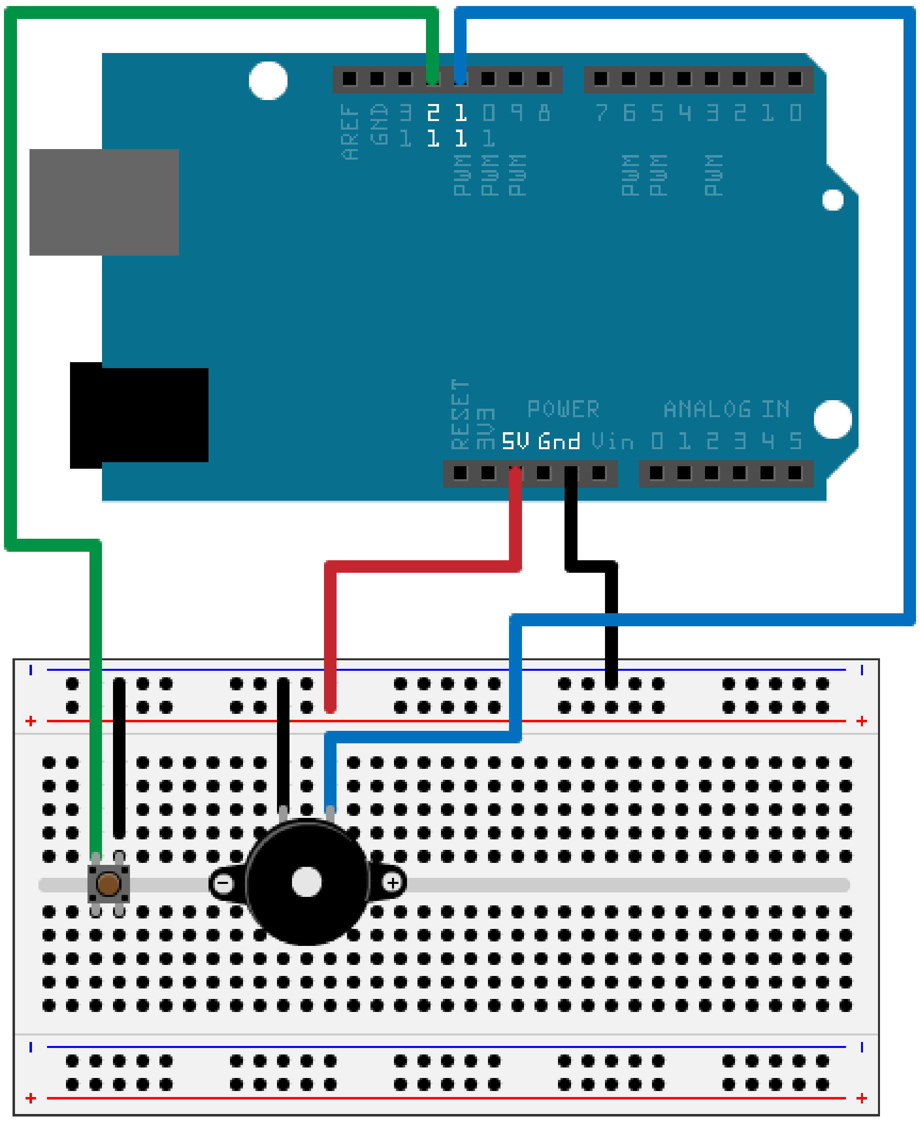
While the code is running, you can rotate the knob on the potentiometer to see how it affects the brightness of the LED.

Control a Piezo speaker using a push button

This part of the example shows how to play a tone controlled by a push button connected to a digital pin on the Arduino hardware. You can also configure a digital pin to `pullup` mode and use the built-in pullup resistor.

1) Connect a Piezo speaker to digital pin 11.

2) Connect a push button to digital pin 12.



To play a tone on the speaker, you can use playTone method to specify the frequency and duration of the sound. Second, the status of a push button can be detected by reading the connected digital pin's value. In order for the push button to work, a pullup resistor needs to be connected to the corresponding digital pin on Arduino board. You can use the built-in pullup resistor by configuring the digital pin mode to `pullup` to enable it. If the button has been pushed, meaning the read back value is `0`, a beep sound is played on the speaker. Execute the following command at the MATLAB prompt to play a sound on the speaker when push button is pressed down.

```
configureDigitalPin(a, 12, 'pullup');
time = 200;
while time > 0
   speaker_status = readDigitalPin(a, 12);
   if speaker_status == 0
       playTone(a, 11, 1200, 1);
```

```
        else
            % Change duration to zero to mute the speaker
            playTone(a, 11, 1200, 0);
        end

       time = time - 1;
       pause(0.1);
   end
```

Clean up

Once the connection is no longer needed, clear the arduino object.

```
clear a
```

Measure Temperature From I2C Device on Arduino® Hardware

This example shows how to use the MATLAB® Support Package for Arduino® Hardware and the I2C interface to communicate with I2C devices.

Contents

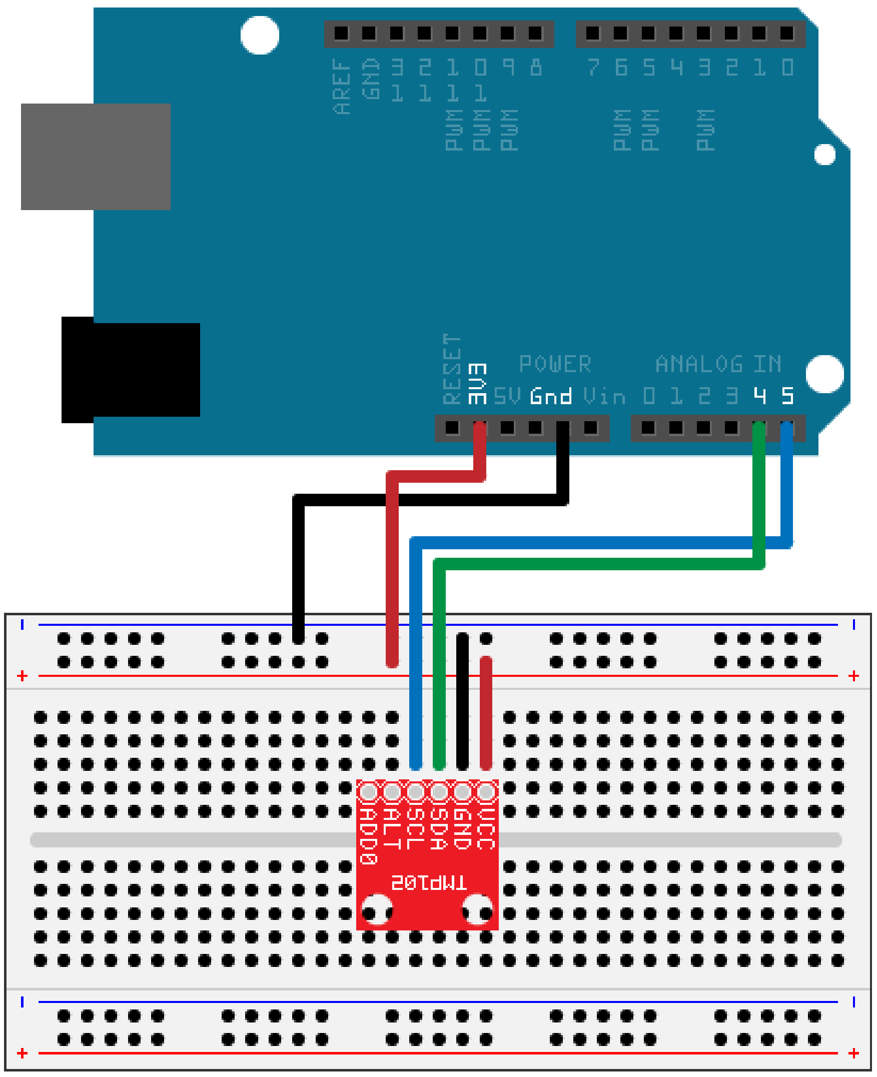
Overview of TMP102 temperature sensor

This example uses TMP102, a two-wire serial output digital sensor, which can read temperature up to a resolution of 0.0625 degree in Celsius. You can also read data from the device in Extended mode with higher measurement limit.

Hardware setup

1) Connect the SDA, SCL, GND and V+ pins of the sensor to the corresponding pins on Arduino hardware. This examples connects SDA and SCL pins to A4 and A5 on Arduino Uno board. If you are using a different board, check the correct pins before connection.

2) Securely connect the power line of the I2C sensor.



Create an I2C device object

1) Create an arduino object and include the I2C library.

```
a = arduino();
```

Or, you can explicitly specify it in the Libraries Name-Value pair at creation of arduino object.

```
a = arduino('com22', 'uno', 'Libraries', 'I2C');
```

2) Scan for available I2C addresses.

```
addrs = scanI2CBus(a)
addrs =

    '0x48'
    '0x60'
```

Note the address of the temperature sensor. You will use it to create the I2C device object.

3) Create the I2C device object

```
tmp102 = i2cdev(a, '0x48')
```

```
tmp102 = 

  i2cdev with properties:

        Pins: A4(SDA), A5(SCL)
         Bus: 0
     Address: 72 (0x48)
```

The bus defaults to `0`. If you are using the dedicated I2C interfaces(SDA1, SCL1) on Due board, for example, make sure to set bus to `1`.

Read temperature value

The sensor's temperature reading is digitized into 12 bits in Normal mode with 8 bits in MSB and 4 bits in LSB. Each LSB equals 0.0625 degrees in Celsius. Write the register address to read from first and then read two bytes of data from it. Use `uint8` data type.

```
write(tmp102, 0, 'uint8');
data = read(tmp102, 2, 'uint8');
temperature = (double(bitshift(int16(data(1)), 4)) + double(bitshift(int16(data(2)), -4)))
 * 0.0625
```

```
temperature =

   23.1875
```

Read temperature with higher measurement limit

With the TMP102 sensor's extended mode, you can measure temperature above 128 degrees by using 13 bits. To do so, you need to write value '60B0' in hex to configuration register at address 1.

```
writeRegister(tmp102, 1, hex2dec('60B0'), 'uint16');
```

Read the temperature from the register to get a more precise result. The TMP102's conversion rate defaults to 4Hz. Hence, pause MATLAB for about 0.25s before each reading.

```
pause(0.25);
data = readRegister(tmp102, 0, 'uint16');
temperature = double(bitshift(bitand(data, hex2dec('FFF8')), -3)) * 0.0625
```

```
temperature =

   23.1875
```

To change back the default configuration, type

```
writeRegister(tmp102, 1, hex2dec('60A0'), 'uint16');
```

Clean up

Once the connection is no longer needed, clear the associate object.

```
clear tmp102 a
```

Communicate with SPI Device on Arduino® Hardware

This example shows how to use the MATLAB® Support Package for Arduino® Hardware to use SPI interface to communicate with MCP42010 Digital Potentiometer.

Contents

Overview of MCP42010 Digital Potentiometer

The MCP42010 device is a 256-position 10KOhm potentiometer SPI device with two independent channels.

It has channel 1 on pin 5(PB1), 6(PW1) and 7(PA1), and also channel 0 on pin 10(PB0), 9(PW0), 8(PA0). Pin 6 and pin 9 are wiper pins. This example uses CS, SCK, SI, SO, VDD, VSS, PB1, PW1 and PA1.

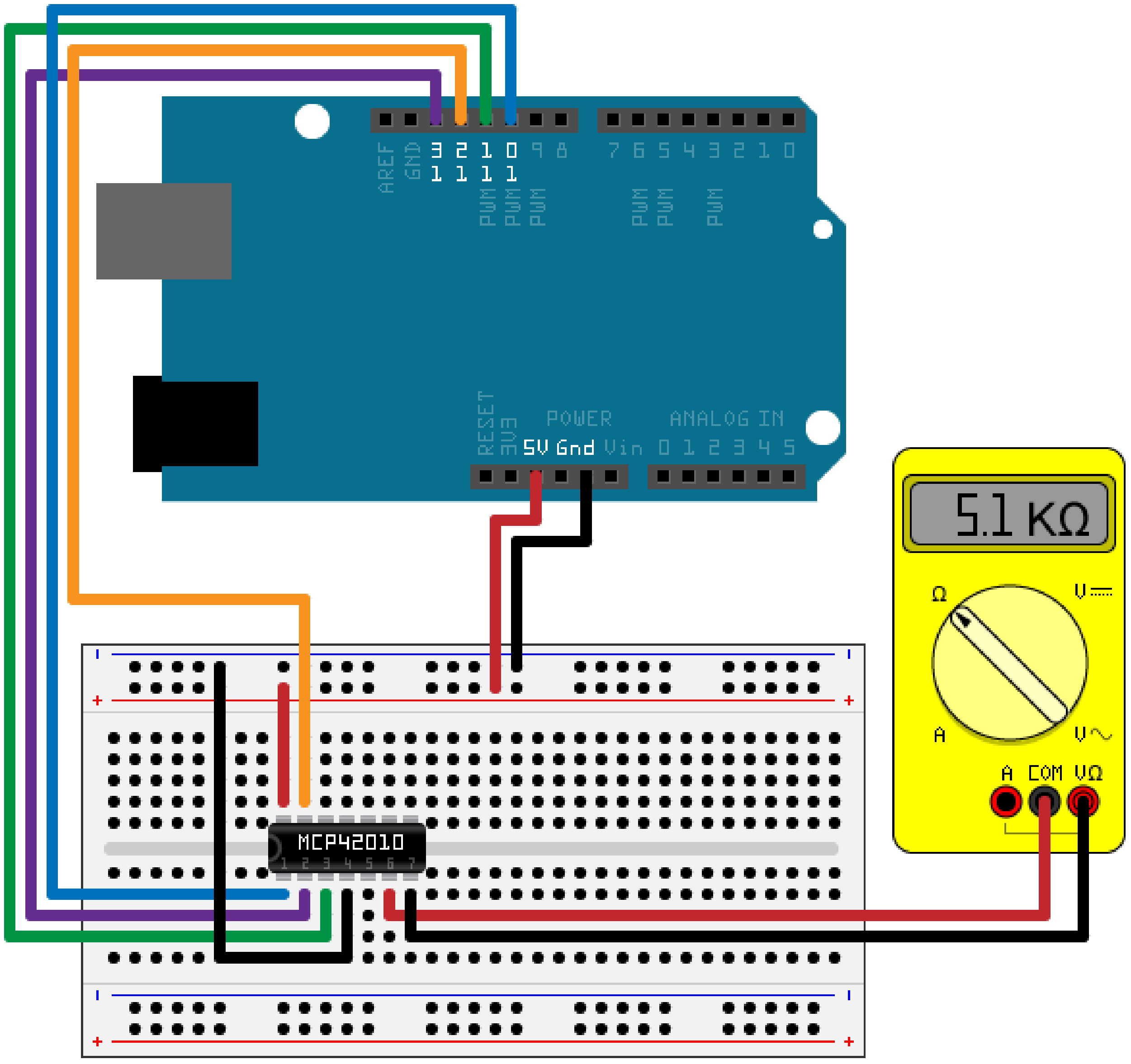
Hardware setup

1) Connect the SI, SO, SCK, CS, VDD and VSS pins of a MCP42010 10KOhm digital potentiometer to the Arduino hardware. This example uses an Arduino Uno board with the following connection.

- SI(MOSI) - digital pin 11
- SO(MISO) - digital pin 12
- SCK - digital pin 13
- CS - digital pin 10
- VDD - 5V
- VSS - GND

If you are using a different board, make sure you connect to the correct pins.

2) Connect a multimeter to PA1 and PW1 to measure the resistance.



Control the digital potentiometer

Create an arduino object and include the SPI library.

```
a = arduino();
```

Or, you can explicitly specify it in the Libraries Name-Value pair at creation of arduino object.

```
a = arduino('com22', 'uno', 'Libraries', 'SPI');
```

Create an spidev object and specify the pin number for chip select.

```
d_pot = spidev(a, 10);
```

Send two bytes of data to change the resistance. Since we are controlling channel 1, the first byte should be 0b00010010 which is 12 in hex. The second byte is the new register data in the range of 0 and 255. The following commands change the resistance of the potentiometer gradually.

```
Rab = 10*1000;
Rw = 52;     % actual wiper resistance
for regVal = 0:50:250
    pot_resistance = Rab*regVal/256+Rw;
    writeRead(d_pot, [hex2dec('12'), regVal], 'uint8');
    fprintf('Current resistance is %d Ohm\n', pot_resistance);
    pause(2);
end
```

```
Current resistance is 52 Ohm
Current resistance is 2.005125e+03 Ohm
Current resistance is 3.958250e+03 Ohm
Current resistance is 5.911375e+03 Ohm
Current resistance is 7.864500e+03 Ohm
Current resistance is 9.817625e+03 Ohm
```

The code runs and displays the readings of the potentiometer.

Clean up

Once the connection is no longer needed, clear the associate object.

```
clear d_pot a
```

Control Servo Motors

This example shows how to use the MATLAB® Support Package for Arduino® Hardware to control a hobby servo motor.

Contents

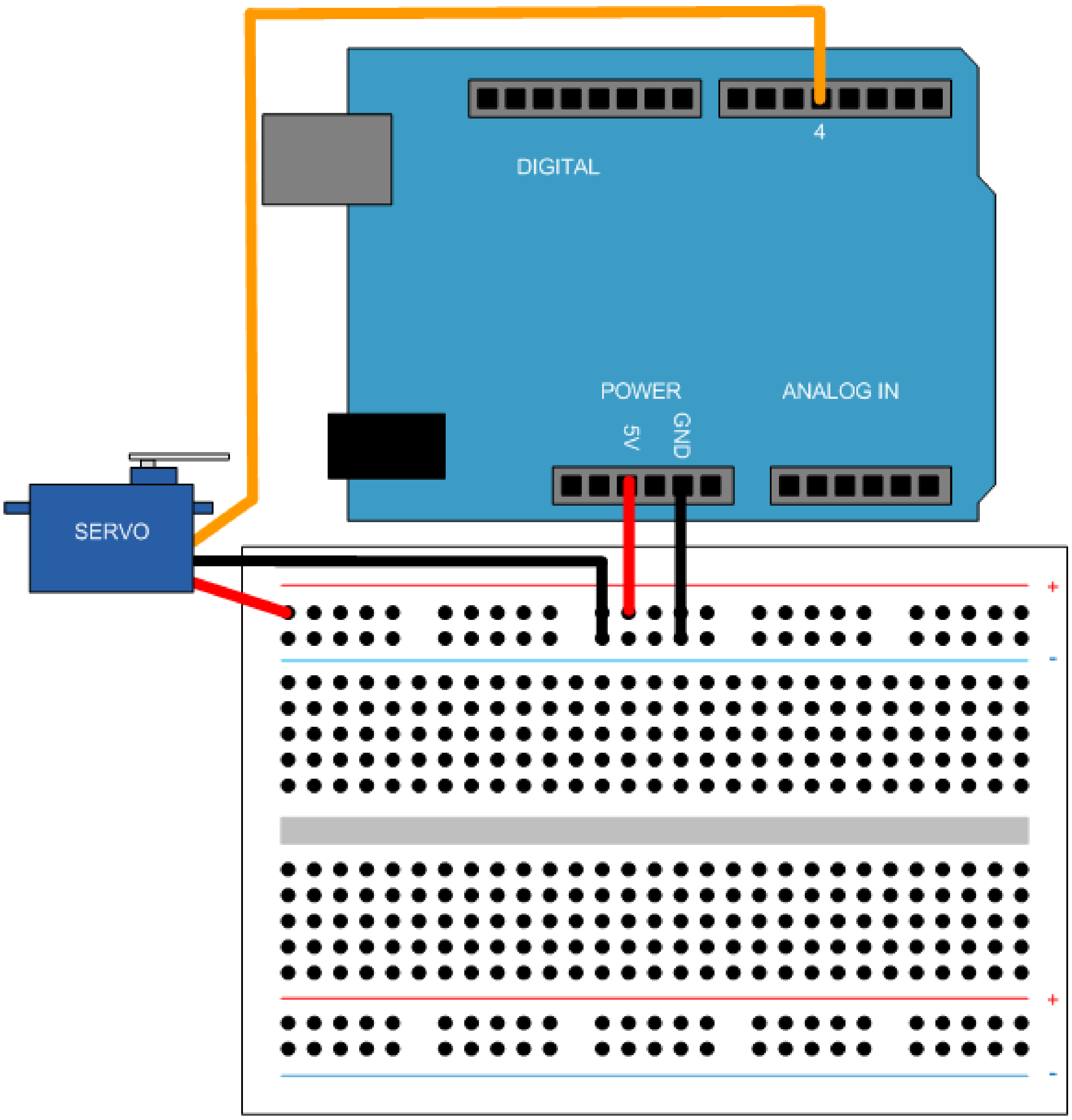
Hardware setup

- Connect an FS5106B servo motor to Arduino hardware,

1. Connect the power wire (usually red) to the 5V pin.

2. Connect the ground wire (usually black) to the ground pin.

3. Connect the signal wire (usually orange) to digital pin 4.



Create servo object and calibrate the motor

Create an arduino object and include the Servo library.

```
    a = arduino();
```

Or, you can explicitly specify it in the Libraries Name-Value pair at creation of arduino object.

```
a = arduino('com22', 'uno', 'Libraries', 'Servo');
```

Create a Servo object.

```
    s = servo(a, 4)
```

```
s =

  Servo with properties:

                 Pins: 4
    MinPulseDuration: 5.44e-04 (s)
    MaxPulseDuration: 2.40e-03 (s)
```

Check your servo motor's data sheet pulse width range values to calibrate the motor to rotate in expected range. This example uses 700*10^6 and 2300*10^-6 for the motor to move from 0 to 180 degrees.

```
clear s;
s = servo(a, 4, 'MinPulseDuration', 700*10^-6, 'MaxPulseDuration', 2300*10^-6)
```

```
s =

  Servo with properties:

                 Pins: 4
    MinPulseDuration: 7.00e-04 (s)
    MaxPulseDuration: 2.30e-03 (s)
```

Write and read Servo position

Change the shaft position of the servo motor from 0(minimum) to 1(maximum) with 0.2, e.g 36 degrees, increment. Display the current position each time the position changes.

```
for angle = 0:0.2:1
    writePosition(s, angle);
    current_pos = readPosition(s);
    current_pos = current_pos*180;
    fprintf('Current motor position is %d degrees\n', current_pos);
    pause(2);
end
```

```
Current motor position is 0 degrees
Current motor position is 36 degrees
Current motor position is 72 degrees
Current motor position is 108 degrees
Current motor position is 144 degrees
Current motor position is 180 degrees
```

Clean up

Once the connection is no longer needed, clear the associate object.

```
clear s a
```

Control Motors Using Adafruit® Motor Shield V2

This example shows how to use the MATLAB® Support Package for Arduino® Hardware to control servo motors, DC motors and stepper motors using Adafruit motor shield v2.

Contents

Hardware setup

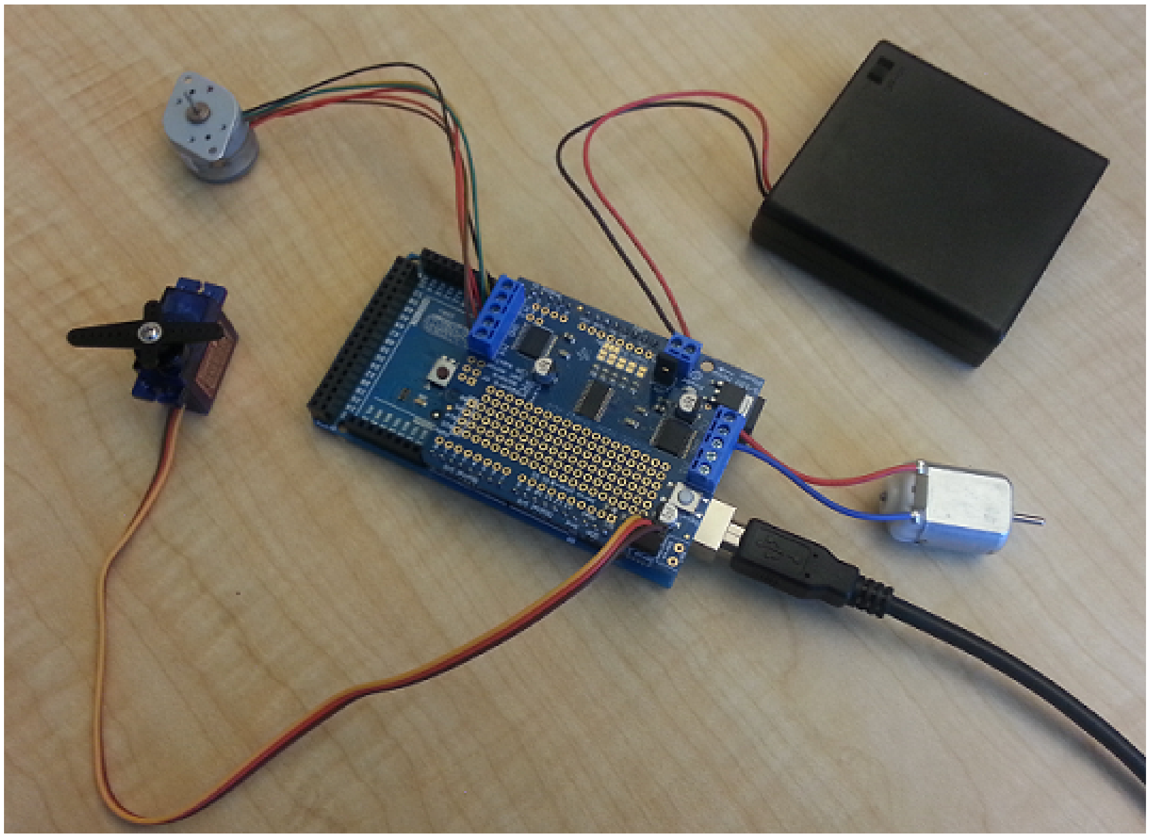
1. Attach Adafruit motor shield to your Arduino hardware.

2. Connect an FS5106B motor to port 1, labeled 'Servo 1' on the shield.

3. Connect a DC toy/hobby motor to port 1, labeled 'M1' on the shield.

4. Connect a six-wire Portescap stepper motor to port 1, labeled 'M1' and 'M2' on the shield. Connect the two middle wires on the stepper motor to the center of the port to ground them together. If you are using four-wire or five-wire stepper motor, check your hardware specs for appropriate connections of each wire.

5. Connect a battery pack that has three AA batteries to the DC jack, labeled with Power and remove the jumper on pins labeled Vin Jumper. This step is optional if your stepper motor does not require a high power supply.



Create shield object

By default, the Adafruit\MotorShieldV2 library is not included in the server code on the board. Create an arduino object and include the Adafruit\MotorShieldV2 library to the hardware.

```
a = arduino('com25', 'uno', 'Libraries', 'Adafruit\MotorShieldV2')
```

```
a =

  arduino with properties:

                   Port: 'COM25'
                  Board: 'Uno'
     AvailableAnalogPins: [0, 1, 2, 3, 4, 5]
    AvailableDigitalPins: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
               Libraries: {'Adafruit/MotorShieldV2', 'I2C', 'Servo'}
```

Create an add-on shield object by specifying the required library name paramter:

```
shield = addon(a, 'Adafruit\MotorShieldV2')
```

```
shield =

  motorshieldv2 with properties:
```

```
             Pins: A4(SDA), A5(SCL)
       I2CAddress: 96 (0x60)
     PWMFrequency: 1600 (Hz)
```

The I2CAddress of a shield is set to `0x60` by default if not specified. Search for available I2C addresses on bus 0 to specify a different address.

```
addrs = scanI2CBus(a,0)
```

```
addrs = 

    '0x60'
    '0x70'
```

Control servo motor on the shield

There are two servo motor ports available on each shield. To create a servo motor object at port 1.

```
s = servo(shield, 1)
```

```
s = 

  Servo with properties:

         MotorNumber: 1
                Pins: 10
    MinPulseDuration: 5.44e-04 (s)
    MaxPulseDuration: 2.40e-03 (s)
```

Set the position of the servo motor's shaft to its maximum position.

```
writePosition(s, 1);
```

See Servo Motor Control example to learn how to use a servo object.

Control DC motor on the shield

There are four DC motor ports available on each shield. Create a DC motor object at port 2.

```
dcm = dcmotor(shield, 2)
```

```
dcm = 
  dcmotorv2 with properties:

    MotorNumber: 2 (M2)
          Speed: 0.00
      IsRunning: 0
```

First, change the motor speed to `0.2`. The sign of the value indicates the direction of the motor rotation that also depends on the wiring of the motor.

```
dcm.Speed = 0.2;
```

Start the motor and change the speed while it is running. Stop the motor when you are done.

```
start(dcm);
dcm.Speed = 0.3;
pause(2);
dcm.Speed = -0.2;
pause(2);
stop(dcm);
```

Control stepper motor on the shield

There are two stepper motor ports available on each shield. To create a stepper motor object at port 2 with 200 steps per revolution.

```
sm = stepper(shield, 2, 200)
```

```
sm =

  stepper with properties:

           MotorNumber: 2
    StepsPerRevolution: 200
                   RPM: 0
              StepType: Single ('Single', 'Double', 'Interleave', 'Microstep')
```

Set the motor's RPM, e.g revolutions per minute, to 10 and move or step the motor 200 steps in one direction and then another 200 steps in the reverse direction.

```
sm.RPM = 10;
move(sm, 200);
pause(2);
move(sm, -200);
release(sm);
```

Clean up

Once the connection is no longer needed, clear the associated object.

```
clear s dcm sm shield a
```