Functionality:

This repo allows users to control an Arduino using serial communication. It supports standard digital/analog pin read/write, servo control and interrupt based encoders (quadrature and incremental).

How To Use:

1)
Navigate to the ServerCode folder and find the code for the type of Arduino you are using. (only Arduino Uno is supported so far). Open the .ino file in the Arduino IDE.
Download the Arduino IDE here.
Upload the code to your Arduino.

2)
Navigate to the ClientCode folder and use the code for the client you are using.

OR

Write your own client code in any language by following the following protocol:

Overview of the Protocol:

Every serial message to the arduino will be a stream of bytes, with a max length of 8 bytes, having the following form:
{function byte, parameter 1, … parameter n, terminator}.

The function byte defines the function to be called on the arduino.
The param bytes are the parameters to the function.
The terminator tells the arduino that the message is over.

The Arduino parses the message, executes the procedure (if possible), and sends a return message of the form: {output, terminator}

If the Arduino encounters an error in executing the procedure, it will return 253 as the output, followed by an error message as a stream of bytes representing ASCII characters

For Arduino functions that return an output, such as digitalRead or analogRead, the output will correspond to the output of the function.

For all other functions, the output will be 0, meaning that the function executed properly.

Example: You want to set up digital pin 1 on the Arduino as digital output and have it output HIGH.

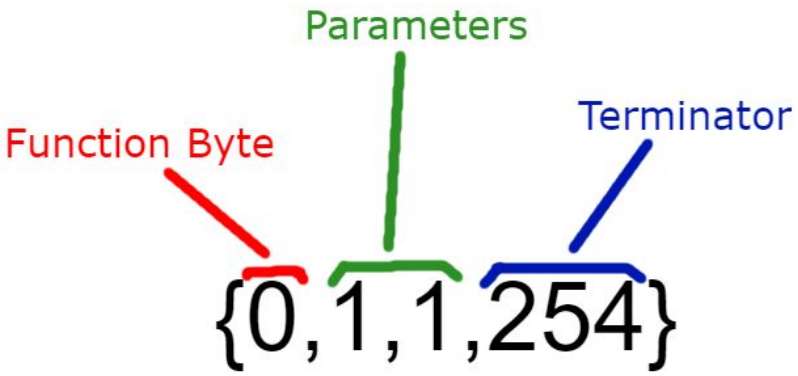
In arduino C++ code that would be done as follows:

```
pinMode(1,OUTPUT);
digitalWrite(1,HIGH);
```

Via the protocol this would be accomplished with the following messages:

{0,1,1,254}
{1,1,1,254}

Each message can be divided into three main sections: The function, the parameters, and the terminator:



The arduino parses the message {0,1,1,254} as follows:

Section	Value	Meaning
Function	0	pinMode
Parameters	1,1	D1,OUTPUT
Terminator	254	End Message

{0,1,1,254} is interpreted as: `pinMode(1,OUTPUT);`

Assuming the arduino successfully runs this command, it will return the following message, meaning that the command ran successfully:

{0,254}

0 signifies a successful completion of the command, and 254 signifies the end of the return message.

Outgoing messages (Client to Arduino)

Special Values:

255 - Null

The max value of a byte, 255, is used by Arduino's Serial.read() function to signify a lack of available serial input. This means any values of 255 will be ignored.

254 - The Terminator

The terminator was chosen as 254. This value signifies the end of a message.

The Function Byte:

Byte Value	Server side function
0	pinMode(pinNumber, mode)
1	digitalwrite(pinNumber, value)
2	analogwrite(pinNumber,value)
3	digitalRead(pinNumber)
4	analogRead(pinNumber)
5	servo(pinNumber)
6	writeServo(pinNumber,value)
7	detachServo(pinNumber)
8	encoder(interruptPin,secondaryPin)
9	getEncoderCount(interruptPin)
10	resetEncoder(interruptPin)
11	detachEncoder(interruptPin)
12	setEncoderDirection(interruptPin,direction)
252	Used internally by the Arduino to represent a serial read timeout. Sending this value as the function byte will return a timeout error.

253	checkConnection(optionalParam)

Function Usage and Parameters:

Pin numbers:
Any function requiring a pinNumber parameter, interprets the byte as follows:

Byte Value	Pin
0	D0
1	D1
2	D2
3	D3
4	D4
5	D5
6	D6
7	D7
8	D8
9	D9
10	D10
11	D11
12	D12
13	D13
14	A0
15	A1
16	A2
17	A3
18	A4
19	A5

pinMode - Function Byte 0

pinMode(pinNumber, pinType)

Sets up [pinNumber] as type [pinType].

pinType

Byte Value	pinType
0	INPUT
1	OUTPUT
2	INPUT_PULLUP

digitalWrite - Function Byte 1

digitalWrite(pinNumber,value)

Writes [value] to [pinNumber]

value

Byte Value	value
0	LOW
1	HIGH

analogWrite - Function Byte 2

analogWrite(pinNumber,value)

Writes [value] to [pinNumber]

[value] corresponds to a duty cycle between 0 (always off) and 252 (always on).

Note: Arduino UNO PWM output pins are 3, 5, 6, 9, 10 and 11. analogWrite will act as digitalWrite for other pins. Values greater than 126 will correspond to HIGH.

digitalRead - Function Byte 3

digitalRead(pinNumber)

Returns the state of [pinNumber], as 0 or 1 (LOW or HIGH) as a single byte

analogRead - Function Byte 4

analogRead(pinNumber)

Returns the state of [pinNumber], as a value between 0 (0v) and 1023 (5v or Aref). Value is returned as a signed int16, which can be interpreted by following the procedure for parsing **Large Signed Return Values.**

Analog input pins are A0 - A5, corresponding to pin numbers 14-19.

servo - Function Byte 5

servo(pinNumber)

Sets up a servo at [pinNumber]

writeServo - Function Byte 6

writeServo(pinNumber,value)

Writes duty cycle, [value] between 0 (minimum duty cycle) and 252 (maximum duty cycle), to servo at [pinNumber].

detachServo - Function Byte 7

detachServo(pinNumber)

Detaches servo at [pinNumber]

encoder - Function Byte 8

encoder(interruptPin,[optional]secondaryPin)

Sets up an encoder with channel A at interruptPin (must be D2 or D3), and channel B at secondaryPin (any other pin). Omitting the secondaryPin will cause the encoder to behave as an incremental encoder

getEncoderCount - Function Byte 9

getEncoderCount(interruptPin)

Returns the count of the encoder at [interruptPin]
Value is returned as a signed int16, which can be interpreted by following the procedure for parsing **Large Signed Return Values.**

resetEncoder - Function Byte 10

resetEncoder(interruptPin)

Sets the count of encoder at [interruptPin] to zero.

detachEncoder - Function Byte 11

detachEncoder(interruptPin)

detaches the encoder at interruptPin.

setEncoderDirection - Function Byte 12

setEncoderDirection(interruptPin,direction)

Sets the direction in which encoder ticks are counted, depending on the value of the direction parameter.

Direction Value	Effect
0	Do not count ticks
1	Count ticks with normal polarity
2	Count ticks with opposite polarity

By default, encoders will count ticks with normal polarity, meaning that if the interrupt channel is **falling**, and the secondary channel is **high**, ticks will increment (and decrement if secondary channel is low). For non-quadrature encoders, positive polarity will always increment ticks on interrupt, and negative polarity will decrement ticks.

checkConnection - Function Byte 253

checkConnection([optional]param)

Takes 1 optional parameter. If this parameter is left out, the arduino will return 0, to signify that the device is working properly.

Including the optional parameter yields the following output from the arduino:

Optional parameter value	Return Value
0	Device info (as a string)
1-253	0

Return messages (Arduino to Client)

Special Values:

254 - The Terminator
Incoming messages use the same terminator, 254, to signify the end of a message.

253 - Error
The Arduino will only return the value 253 if it encounters an error.

Other:

0 - Success
Zero is used to signify the successful completion of a function without a return value, but it is not only reserved for functions without return values. For functions that have return values (such as digital read or analog read) zero can also be returned so signify a specific reading.

Small Return Values:

Functions with small unsigned return values, such as digitalRead, return their output as a single byte.

Large Signed Return Values:

Functions with large or signed return values, such as readEncoder (which returns int16), the value is broken apart into multiple bytes. The first byte is the sign of the value (1 for negative 0 for positive). Subsequent bytes define the absolute value, in little endian format (the earliest byte to send has the smallest value). **Important: Each "byte" in the absolute value is an 8 bit representation of a 7 bit number. To get the actual value from the stream of bytes compute the following:**

$$Value = \sum_{i=0}^{n} (Byte_i) * (2^{7*(i-1)})$$

This was done because at least one possible value of each byte had to be reserved for the terminator. The solution is to sacrifice 1 bit of data per byte. **Note:** this means an int16 returns 3 bytes, and int32 returns 5 bytes, etc.

Example: The arduino returns an int16 value of -300.

The arduino would send the following message:

{1,44,2,0,254}

Which can be separated into the following sections:

Section	Value	Meaning
Sign	1	-1
Absolute Value	44,2,0	$44 * 2^0 + 2 * 2^7 + 0 * 2^{14} = 300$
Terminator	254	End Message

This should be interpreted as -300.

Errors:

Error messages start with the byte value 253, followed by a sequence of bytes representing ASCII characters, and ending in the terminator (254). For example, the error message: “errorMsg1” would be represented by the following sequence of bytes:

{253,101,114,114,111,114,77,115,103,49,254}