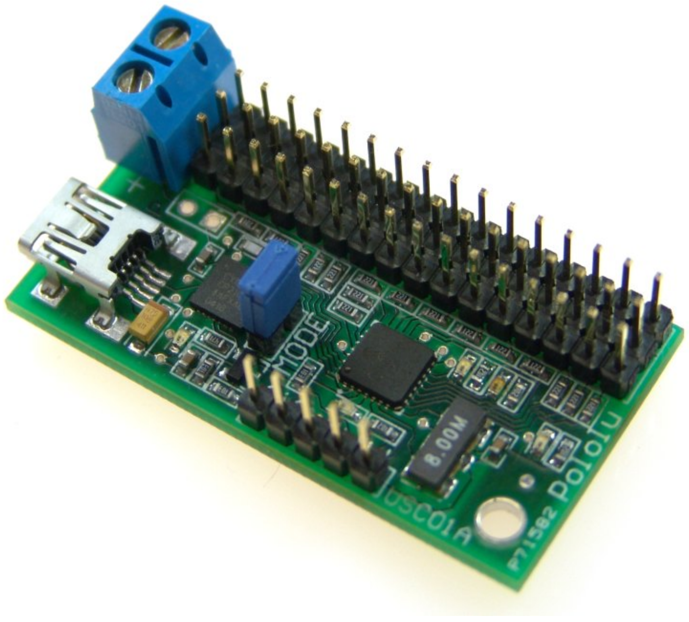
Pololu

USB 16-Servo Controller

User's Guide (preliminary)



Contents:

USC01A

Important Safety Warning

The servo controller module is not intended for young children! Younger users should use this module only under adult supervision. **By using this product, you agree not to hold Pololu liable for any injury or damage related to the use or to the performance of this product. This product is not designed for, and should not be used in, applications where the malfunction of the product could cause injury or damage.**

Contacting Pololu

You can check the Pololu web site at **`http://www.pololu.com/`** for the latest information about the servo controller, including color pictures, application examples, and troubleshooting tips.

We would be delighted to hear from you about your project and about your experience with our motor controller. You can contact us through our online feedback form or by email at `support@pololu.com`. Tell us what we did well, what we could improve, what you would like to see in the future, or anything else you would like to say!

Connecting the Servo Controller

Attention: Before connecting this device to your computer, please follow the instructions and install drivers available at:

http://www.pololu.com/products/pololu/0390/

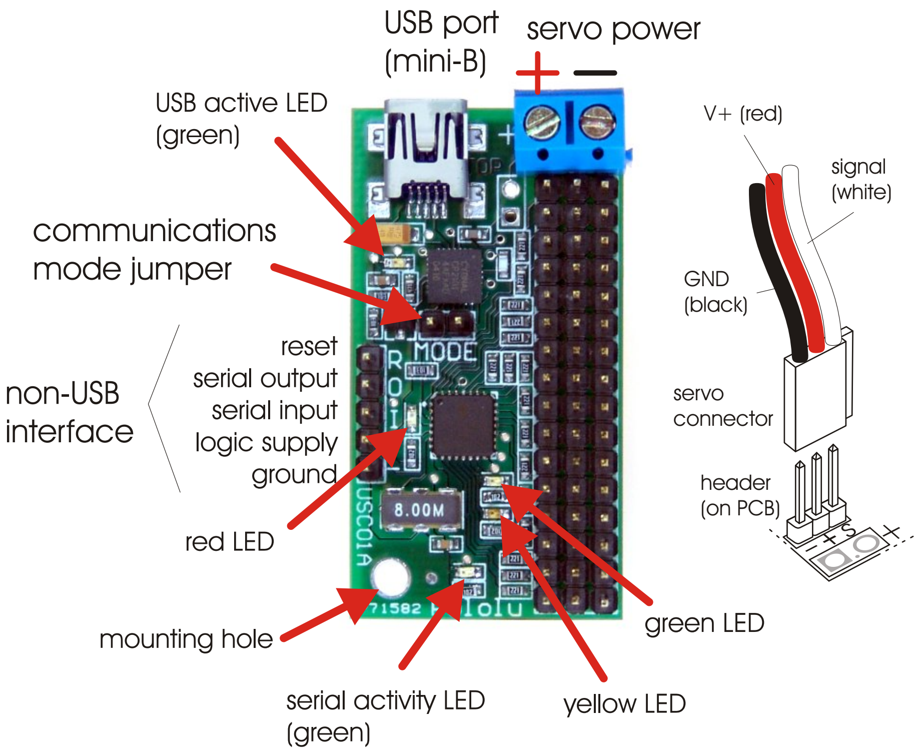
The connections to the servo controller module are shown on the next page. In general, two connections need to be made: a servo power connection, and a control connection. The control connection can be the USB port or the serial interface on the left side of the board. With either control method, apply servo power only after you have established the other connection(s) and verified proper communication with the servo controller.

USB interface. In most cases, the USB interface is used for communicating with the servo controller. After the driver is installed, plug the servo controller into a USB port using an appropriate cable. Follow any on-screen prompts to complete the USB installation. The servo controller will then look like a serial port to the computer, and you can send commands to the servo controller as if you were using a standard serial port. Power for the board is supplied by the USB connection, so no additional power needs to be connected except for the servo supply.

Servo Power. The dedicated servo power power is only for powering the servos, which usually require 4.8-6 V to operate. Keep in mind that the servos can draw a substantial amount of current, so a high-capacity rechargeable battery is usually the best servo supply.

Non-USB Interface. The servo controller can be connected to a TTL-level serial interface instead of a USB port. In that case, a 5V logic supply must also be connected. The pinout of the non-USB interface is shown in the diagram below. If you are using several servo controllers to control more than 16 servos, you can connect just one of them to the USB port, and then connect the logic supply and grounds of the remaining boards together. The O (serial output) pin of the controller connected to the USB port should then be connected to the serial input pins of the other controllers. The reset line is optional; making it low will reset the servo controller.

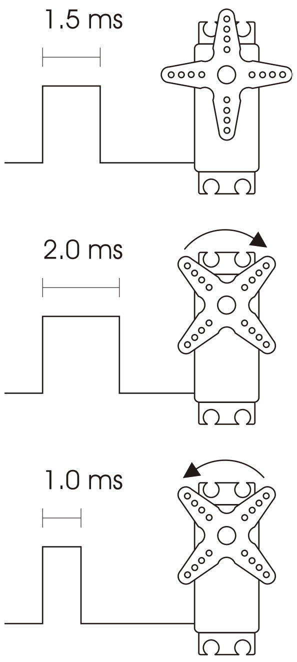
Servos. When connecting servos, be careful because the servo header pins are not polarized. Make sure to connect your servos correctly, or they may be destroyed. The signal (usually white or yellow) wire should be closest to the PIC microcontroller (farthest from the edge of the board), and the black wire should be closest to the edge of the board. Some servo connectors have a polarizing nub that indicates the signal lead; that notch should be on the pin farthest from the edge of the PCB.

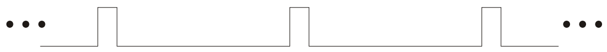


How Servos and the Servo Controller Work

Radio Control (RC) hobby servos are small actuators designed for remotely operating model vehicles such as cars, airplanes, and boats. A servo might typically move the control surface of an airplane or the steering mechanism in a car. A servo contains a small motor and gearbox to do the work, a potentiometer to measure the position of the output gear, and an electronic circuit that controls the motor to make the output gear move to the desired position. Because all of these components are packaged into a compact, low-cost unit, servos are great actuators for robots.

An RC servo has three leads: two for power and ground, and a third for a control signal input. The control signal is a continuous stream of pulses that are 1 to 2 milliseconds long, repeated approximately fifty times per second, as shown below. The width of the pulses determines the position to which the servo moves. The servo moves to its neutral, or middle, position when the signal pulse width is 1.5 ms. As the pulse gets wider, the servo turns one way; if the pulse gets shorter, the servo moves the other way. Typically, a servo will move approximately 90 degrees for a 1 ms change in pulse width, but the exact correspondence between pulse width and servo varies from one servo manufacturer to another.





The Pololu servo controller performs the processor-intensive task of simultaneously generating 16 independent servo control signals. The servo controller can generate pulses from 0.25 ms to 2.75 ms, which is greater than the range of most servos, and which allows for a servo operating range of over 180 degrees.

Internally, the servo controller maintains a servo position value that is two times the pulse width, measured in microseconds. Thus, the 1.5 ms neutral position, which is 1500 microseconds long, is represented internally as 3000. The internal values thus range from 500 to 5500. Various interface modes allow the user to set the position value for each servo in multiple ways, which are described in depth in the "Using the Servo Controller" section.

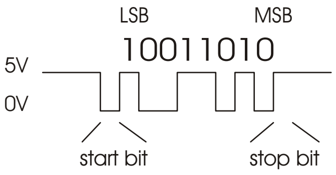
Using the Servo Controller

Initial Power-up

When the servo controller first turns on (or is reset), and the serial input is disconnected (or high), the yellow, red, and green indicator LEDs turn on in sequence. **Do not send any serial data to the servo controller at this time.** The yellow LED should then stay lit, indicating that the servo controller is ready for serial input. If the serial input is low during power up or reset, then all LEDs will be on until the serial line is made high, at which time the yellow LED will turn on, indicating that the servo controller is ready.

Serial Input

The serial commands sent to the servo controller must be sent eight bits at a time, with no parity and one stop bit (sometimes abbreviated 8N1). The serial input on the non-USB input must be *non-inverted*, meaning that a zero is sent as a low voltage, and a one is sent as a high voltage, as shown in the diagram to the right. If the USB port is used, then it is only necessary to configure the PC software to communicate to the servo controller port with the 8N1 settings and at the desired rate. *Commands sent to the serial input* ***must*** *conform to the above format and use the appropriate protocol (described in detail later) or else the servo controller and other devices connected to the serial line may behave unexpectedly.*



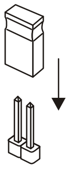
After the servo controller turns on and determines the communication mode (see below), it waits for a serial input to determine the baud rate. If the detected baud rate is too high, the red LED will turn on and the green LED will flash quickly. If the serial rate is too slow, the red LED will turn on and the yellow LED will flash. From this point on, the servo controller behavior depends on the communication mode. **Once you choose a baud rate, all subsequent transmissions must be at that same baud rate.**

Indicator LEDs. The green LED at the bottom of the board indicates serial activity: it should flicker whenever the servo controller receives data. The other green LED indicates servo power: if any servo is on, the LED turns on (in Mini SSC II mode, it is always on). The yellow LED indicates a warning regarding position: either the absolute or neutral position you have requested is out of range, or a combination of neutral, range, and 7-bit or 8-bit position caused the internal position variable to go out of range. The position will just be limited to the max or min, and the yellow LED will go out when all requested positions are in range. The red LED indicates a fatal error that prevents further operation.

Interface Options
You can communicate with the servo controller using one of two communication protocols. One of the two interface modes is chosen based on the state of the "MODE" jumper when the servo controller is powered up; you cannot change modes without resetting the servo controller or turning it off and then on.

Pololu Mode: The default mode, when the mode jumper is open (no shorting block), is a Pololu protocol used for controlling multiple serial devices. In this mode, the servo controller can be on the same serial line as other devices such as our Dual Serial Motor Controller. This mode also allows access to all of the special features of the servo controller, such as setting speeds, ranges, and neutral settings.

Mini SSC II Mode: This mode is set by placing the shorting block over the two mode pins. This setting allows the servo controller to respond to the protocol used by the Mini SSC II servo controller made by Scott Edwards Electronics. This protocol is more simple, but it only allows the user to specify the desired servo positions in only one way. In this mode, the servo controller is not compatible with other Pololu serial peripheral products.



Mini SSC II Mode
Baud Rate. The available baud rate range in this mode is approximately 500-10k baud, but the Mini SSC II only works at 2400 or 9600 baud. If you want to put a Mini SSC II servo controller on the same serial line as your Pololu 16-servo controller, you must use one of the two baud rates that the Mini SSC II can support.

Protocol. To set the servo position, send a sequence of three bytes. The first byte is a synchronization value that must always be 255. Byte 2 is the servo number, and it must not be 255. Byte 3 is the position to which you want the servo to move.

start byte = 0xFF	servo number, 0x00-0xFE	Servo position, 0x00-0xFE

Two motion ranges are available in this mode. Each Pololu servo controller responds to 32 servo numbers. Addressing the lower 16 will move them within an approximately 90 degree range, while addressing the upper 16 servo numbers will give twice the range. For example, sending the command sequence [255, 18, 254] will move servo 2 all the way to one extreme of its range in 180-degree mode. The servo controller can be configured to respond to different sets of servo numbers (see page 9). The servo controller is shipped with the default servo numbers of 0-31, but you can set a controller to respond to numbers 32-63, 64-95, and so on to control larger quantities of servos through one serial line or USB port. If you send servo numbers that are not recognized, the servo controller will ignore the command. Up to eight servo controllers can be connected on one serial line to independently control up to 128 servos.

Pololu Mode

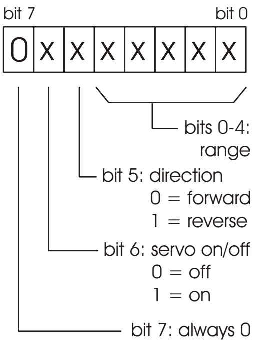
In this mode, there are several options for controlling your servos. As mentioned in the "How Servos and the Servo Controller Work" section, the servo controller holds an internal variable for each servo, the value of which ranges from 500 to 5500, where the number corresponds to the pulse length in increments of half of a microsecond. The various commands deal with setting these internal values. With *absolute* commands, you simply set the value for each servo. In 7- and 8-bit modes, you set *neutral*, *range*, and *direction* parameters for each servo; then, when you send a 7- or 8-bit position command, the servo controller combines all of the parameters to obtain the actual servo position. Whether you are in absolute mode or not, you can individually control the speed of each servo and whether the servo is on or not (most servos shut off when they receive no pulses). The following section describes the interface details.

Baud Rate. The available baud rate range in this mode is approximately 2,000-40,000 baud. It is possible for the servo controller to operate at rates as high as 57600 baud, but the success of the attempt will depend on the exact speed of your serial control source. The servo controller is not committed to standard baud rates, so if you have a device that can transmit at a maximum rate of 45,000 baud, try it, and it might work.

Protocol. To communicate with the servo controller, send sequences of five or six bytes. The first byte is a synchronization value that must *always* be 0x80 (128). Byte 2 is the Pololu device type number, which is 0x01 for the 16-servo controller. Byte 3 is one of six values for different commands to the controller; the commands are discussed below. Byte 4 is the servo to which the command should apply. Bytes 5 and possibly 6 are the data values for the given command. **In every byte except the start byte, bit seven must be clear. Thus, the range of values for bytes 2-6 is 0-0x7F (0-127).**

start byte = 0x80	device ID = 0x01	command	servo num	data 1	data 2

Command 0: Set Parameters (1 data byte)



- Bit 6 specifies whether a servo is on or not; a 1 turns the servo on, and a 0 (default) turns it off.
- Bit 5 sets the direction the servo moves, which only applies to 7- and 8-bit position commands. If the bit is 0 (default), a larger position number causes the output pulse to get bigger; if the bit is 1, a larger position number will make the output pulse shorter.
- Bits 0-4 set the range through which the servo moves in 7- and 8-bit commands. A larger value will give a larger range, and setting the range to 0 will make the servo always stay at neutral. Given the same range setting, an 8-bit command will move the servo through twice the range of a 7-bit command. The default range setting is 15, which will give approximately 180 degrees in 8-bit commands and 90 degrees in 7-bit commands.

Command 1: Set Speed (1 data byte)
This command allows you to set the speed at which the servo moves. If the speed is set to 0 (default), the output pulse will instantly change to the set position. If the speed value is nonzero, the pulse changes gradually from the old position to the new position. With a speed of 1, the pulse width changes at 50 microseconds per second, up to a maximum speed of 6.35 ms per second with a speed setting of 127.

Command 2: Set Position, 7-bit (1 data byte)
When this command is sent, the data value is multiplied by the range setting for the corresponding servo and adjusted for the neutral setting. This command can be useful in speeding up communications since only 5 total bytes are sent to set a position. Setting a servo position will automatically turn it on.

Command 3: Set Position, 8-bit (2 data bytes)
This command is just like the 7-bit version, except that two data bytes must be sent to transfer 8 bits. Bit 0 of data 1 contains the most significant bit (bit 7 of your position byte), and the lower bits of data 2 contain the lower seven bits of your position byte. (Bit 7 in data bytes sent over the serial line must always be 0.)

Command 4: Set Position, Absolute (2 data bytes)
This command allows direct control of the internal servo position variable. Neutral, range, and direction settings do not affect this command. Data 2 contains the lower 7 bits, and Data 1 contains the upper bits. The range of valid values is 500 through 5500. Setting a servo position will automatically turn it on.

Command 5: Set Neutral (2 data bytes)
Setting neutral only applies to 7- and 8-bit commands. The neutral value sets the middle of a range, and corresponds to a 7-bit position value of 63.5 or an 8-bit position value of 127.5. The neutral position is an absolute position just like command 4, and setting the neutral position will move the servo to that position. The default value is 3000. It may be useful to change neutral if you change servos and need to calibrate your system, or if you cannot get your mechanical linkages to just the right lengths.

Tip: Setting neutral and servo direction can be useful if you have a device, such as a walking robot, that has multiple symmetrical structures on two sides of a chassis. Instead of determining a sequence of positions for each leg individually, you can design a single leg, and then use the same position values for other legs, changing only the neutral position and direction as necessary.

Setting and Checking the Servo Numbers

The USB 16-servo controller has the convenient feature of allowing the user to set the servo numbers to which the controller responds. By default, the servo controller responds to servo numbers 0-15 (in Pololu mode), but you can set it to respond to numbers 16-31, 32-47, all the way to 112-127. (In Mini SSC II mode, the servo controller would respond to numbers 0-31, 32-63, all the way through 224-254.) This feature is useful if you want to use more than one servo controller at a time to control up to 128 independent servos.

To set the servo numbers, put the servo controller in Pololu mode (shorting block removed from J1) and send the serial sequence [128, 2, <*servonums*>], where <*servonums*> is a number from 0 through 7. A setting of 0 will make the servo controller respond to servo numbers 0-15 (in Pololu mode), a setting of 1 will make it respond to servo numbers 16-31, and so on.

start byte = 0x80	change servo numbers = 0x02	new setting, 0x00-0x08

Upon receiving the command, the servo controller will turn on the red and yellow LEDs and quickly flash the green LED <*servonums*> + 1 times. The green LED will thus quickly flash 1-8 times. The green LED will then pause for approximately 1 second before flashing again. The servo controller must be reset (or power turned off and back on) before it can be used.

If you want to just see the servo numbers setting without changing it, use the above command, but use the value 8 for <*servonums*>. The servo number settings will remain unchanged, but the green LED will flash to indicate the servo numbers, as described above.

Example of Using the Servo Controller with a BASIC Stamp II

To use the Pololu servo controller with the BASIC Stamp 2, use the “serout” command. The command has several options that are explained in the BASIC Stamp manual, but only the pin, baudmode, and output data need to be specified for use with the servo controller.

In the *Pololu mode* example on page 11, servo 0 is set to its lowest speed, and then servo 0 and 1 are moved back and forth with 10 second pauses to show the difference in speeds. In the example, the servo controller serial line is connected to I/O pin 15 of the BASIC Stamp. The second argument, “84”, sets the baud rate to 9600 and sets other parameters to 8 bits, no parity, and non-inverted mode. The arguments in the square brackets are the values actually sent over the serial line. The sync byte, $80 (the $ sign indicates hex; you could also write 128), and the second byte, $01, are always necessary for controlling the servo controller. The third byte is the command number, where 1 is the “set speed” command and 4 is the “set absolute position” command. The last value (or 2 values) is the argument for the command. The data value ‘1’ in the first line sets the speed to 1. The 13 and 127 set the position to (13*128 + 127) = 1791, which corresponds to output pulses of about 0.9 ms.

In mini SSC II mode (with shorting block on mode jumper), the commands all take the form,

```
serout 15, 84, [255, <servo-num>, <servo-pos>]
```

where <servo-num> is the servo number, from 0-15, and <servo-pos> is the servo position, from 0-254.

BS2 pin to use
for serial out

Serial settings
396: 2400 baud
84: 9600 baud
6: 38400 baud

Required header
for SSC01A

Command number:
(01 is set speed)

Servo number
for command

Data value
for command

```
    serout 15,84,[$80,$01,$01,0,1]     'set servo 0 to lowest speed


testloop:
    serout 15,84,[$80,$01,$04,0,13,127]     'set pos servo 0
    serout 15,84,[$80,$01,$04,1,13,127]     'set pos servo 1

    pause 10000
    serout 15,84,[$80,$01,$04,0,35,127]
    serout 15,84,[$80,$01,$04,1,35,127]

    pause 10000
    goto testloop
```

The Pololu USB 16-Servo Controller

With the Pololu USB 16-servo controller, you can control up to sixteen RC servos via a USB port or with its asynchronous serial (UART) interface. When installed, the USB servo controller appears as a serial port to the host computer. Programming is therefore as easy as sending commands to a serial port, and as an added benefit, the servo controller is compatible with many existing programs. The servo controller can be connected to additional serial devices, such as our serial motor controllers, and thereby function as both a servo controller and a USB-to-UART bridge.

With its dual USB and UART interface, the servo controller can at first be used with a PC to quickly develop motion sequences with the advantage of graphical interfaces and quick program changes. For projects without dedicated PCs, such as small autonomous robots, the final motion sequences can then be transferred to an embedded controller, which can communicate with the servo controller over the standard asynchronous serial connection.

Specifications

PCB size................................	1.0" x 1.9" (including USB connector)
Number of servo ports............	16
Pulse width range...................	0.25-2.75 ms
Resolution..............................	0.5 microsecond (about 0.05 degree)
Supply voltage........................	5 V
I/O voltage.............................	0 and 5 V
Serial baud rate.......................	1200-38400 (automatically detected)
Current consumption..............	40 mA (average)