

M17 Series Servomotors - DATASHEET



Affordable and Simple All-in-One Motion Control
From Education to Innovation



Introduction

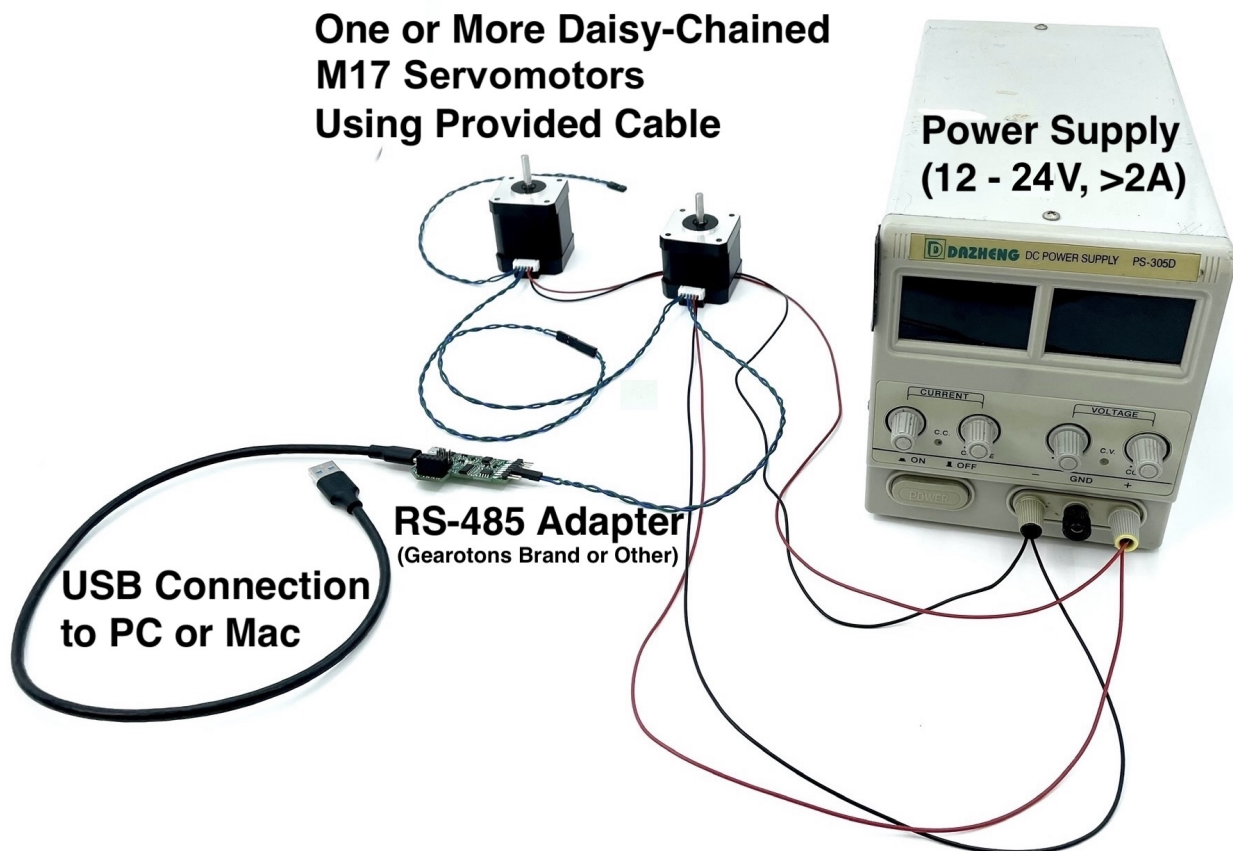
The M17 Series Servomotors are all-in-one motion control solutions that integrate a motor, motor driver, motion controller, and encoder in a single compact package. These servomotors feature a RS-485 communication interface, enabling multiple units to be daisy-chained together and controlled from a single connection point. Available in three models - M17-60, M17-48, and M17-40 - the series offers flexible options to match specific torque requirements while maintaining consistent control characteristics. Each M17 Series Servomotor comes with sophisticated control features including multiple operation modes, self-calibration capabilities, and built-in status monitoring through LED indicators. The motors can be easily controlled from any platform including Mac, PC, Raspberry Pi, or Arduino (requiring only a low-cost RS485 adapter), making them ideal for both educational and industrial applications. The series supports precise position control with trapezoid movement profiles, closed-loop control mode, and comprehensive error handling. With standardized NEMA 17 mounting dimensions, wide voltage range (12-24V), and robust communication protocol, M17 Series Servomotors provide a reliable and flexible solution for applications requiring precise motion control, from robotics and CNC machines to automated testing equipment and scientific instruments.

Key Features

- High level of integration combines a motor, motor driver, motion control system, and RS-485 communication interface
- Compact form factor nearly the same size as a NEMA 17 stepper motor with the same specifications
- Standardized NEMA 17 mounting dimensions
- Wide voltage range (12-24V) for flexible power options
- High-precision position control with built-in encoder and PID control loop that runs at 32 kHz
- Integrated over-current, over-voltage, and over-temperature protection

- High maximum speed can reach 580 RPM
- Torque-to-weight ratio is the same as an equivalent stepper motor
- Compatible with a wide variety of interfaces and hardware, such as Raspberry Pi, Arduino, ESP32, Mac, and PC
- We strive to provide excellent documentation and tutorials to get you up and running fast
- We provide AI friendly documentation in case you want your favorite AI to do all the work for you
- Suitable for robotics, CNC, automation, scientific instruments, testing jigs, 3D printers, and everything else
- Available in different sizes so you can find the right torque and price for your application

Connection Diagram



Unit System

The M17 Series Servomotors have certain internal units so that they can perform the calculations associated with motion efficiently (using integer math). It is the responsibility of the controlling software to support multiple units of measurement for various quantities. Our Python and Arduino libraries handle unit conversions automatically, allowing you to work with your preferred units. Below are the supported units for each quantity:

| Quantity | Available Units |
|--------------|--|
| Time | timesteps, seconds, milliseconds, minutes |
| Position | shaft rotations, degrees, radians, encoder counts |
| Velocity | rotations per second, rpm, degrees per second, radians per second, counts per second, counts per timestep |
| Acceleration | rotations per second squared, rpm per second, degrees per second squared, radians per second squared, counts per second squared, counts per timestep squared |
| Current | internal current units, milliamps, amps |
| Voltage | millivolts, volts |
| Temperature | celsius, fahrenheit, kelvin |

Getting Started Guide

To help you get started with your M17 Series Servomotor, we provide a comprehensive online guide that covers everything from initial setup to advanced protocol implementations. This guide includes:

- Step-by-step setup instructions
- Detailed communication protocol documentation
- Programming examples and code snippets
- Description of error codes
- Troubleshooting tips and best practices



[Click Here to Visit our Getting Started Guide](#)

Indicator LEDs and Buttons

The servomotor has two status LEDs (Green and Red). The green LED flashes slowly to show a heart beat and quickly to indicate that the bootloader is running rather than the application. The red LED will light up briefly to show communication on the bus and will indicate fatal error codes by flashing a certain number of times.

The servomotor has two buttons labelled "Reset" and "Test". The Reset button will reset the internal microcontroller and all state will go back to default values. The Test button will cause the motor to spin. Press briefly to let it spin one way and press for more than 0.3 seconds and release to let it spin the other way. Hold down for at least 2 seconds and release to cause the motor to go to closed loop mode. Hold down for more than 15 seconds and release to let the motor perform a calibration on itself. Note that it will spin during calibration and must be able to spin freely for calibration to be successful, so remove any loads before doing this operation.

Communication Protocol Overview

The M17 Series Servomotors use RS-485 for communication, enabling multiple motors to be daisy-chained together. Each motor has a factory assigned 64-bit unique address so that it can be addressed individually on an RS-485 bus. Each motor can also be assigned a unique 1-byte alias (0-251) in order to save bytes in the communication packet and still get similar individual control. There is also a broadcast alias, which is 255. The protocol supports a comprehensive set of commands for motion control, configuration, and status monitoring. Firmware update through the RS-485 interface is supported. Communication baud rate is fixed at 230400.

Command Reference Summary

For the up to date source of truth for all available commands, you can look at this document.



https://github.com/tomrodinger/servomotor/blob/main/python_programs/servomotor/motor_commands.json

You can also run this command:

```
pip3 install servomotor # run this just once to install the library and programs
motor_command.py -c
```

This will print out the information contained in the motor_commands.json file in a nicer way and give some usage information for sending commands to the motor from the command line.

Basic Control

| Command | Description |
|-----------------|--|
| Disable MOSFETs | Disables the MOSFETs (note that MOSFETs are disabled after initial power on). |
| Enable MOSFETs | Enables the MOSFETs. |
| Reset time | Resets the absolute time to zero (call this first before issuing any movement commands) |
| Emergency stop | Emergency stop (stop all movement, disable MOSFETs, clear the queue) |
| Zero position | Make the current position the position zero (origin) |
| System reset | System reset / go to the bootloader. The motor will reset immediately and will enter the bootloader. If there is no command sent within a short time, the motor will exit the bootloader and run the application from the beginning. |

Motion Control

| Command | Description |
|------------------------|---|
| Trapezoid move | Move immediately to the given position using the currently set speed (the speed is set by a separate command) |
| Go to position | Move to this new given position in the amount of time specified. Acceleration and deceleration will be applied to make the move smooth. |
| Homing | Homing (or in other words, move until a crash and then stop immediately) |
| Go to closed loop | Go to closed loop position control mode |
| Move with acceleration | Rotates the motor with the specified acceleration |
| Move with velocity | Rotates the motor with the specified velocity. |
| Multimove | The multimove command allows you to compose multiple moves one after another. Please note that when the queue becomes empty after all the moves are executed and the motor is not at a standstill then a fatal error will be triggered. |

Configuration

| Command | Description |
|--------------------------------------|---|
| Set maximum velocity | Sets maximum velocity (this is not used at this time) |
| Set maximum acceleration | Sets max acceleration |
| Start calibration | Starts a calibration, which will determine the average values of the hall sensors and will determine if they are working correctly |
| Set maximum motor current | Set the maximum motor current and maximum regeneration current. The values are stored in non-volatile memory and survive a reset. |
| Set safety limits | Set safety limits (to prevent motion from exceeding set bounds) |
| Test mode | Set or trigger a certain test mode. This is a bit undocumented at the moment. Don't use this unless you are a developer working on test cases. |
| Set PID constants | Set PID constants for the control loop that will try to maintain the motion trajectory. |
| Set max allowable position deviation | Set the amount of microsteps that the actual motor position (as measured by the hall sensors) is allowed to deviate from the desired position. Throw a fatal error if this is exceeded. |
| CRC32 control | Enable or disable CRC32 checking for commands |

Status & Monitoring

| Command | Description |
|--------------------------------|---|
| Get current time | Gets the current absolute time |
| Get n queued items | Get the number of items currently in the movement queue (if this gets too large, don't queue any more movement commands) |
| Get hall sensor position | Get the position as measured by the hall sensors (this should be the actual position of the motor and if everything is ok then it will be about the same as the desired position) |
| Get status | Gets the status of the motor |
| Control hall sensor statistics | Turn on or off the gathering of statistics for the hall sensors and reset the statistics |
| Get hall sensor statistics | Read back the statistics gathered from the hall sensors. Useful for checking the hall sensor health and noise in the system. |
| Get position | Get the current desired position (which may differ a bit from the actual position as measured by the hall sensors) |
| Get comprehensive position | Get the desired motor position, hall sensor position, and external encoder position all in one shot |
| Get supply voltage | Get the measured voltage of the power supply. |
| Get max PID error | Get the minimum and maximum error value observed in the PID control loop since the last read. |
| Get temperature | Get the measured temperature of the motor. |
| Get debug values | Get debug values including motor control parameters, profiler times, hall sensor data, and other diagnostic information. |
| Get communication statistics | Get and optionally reset the CRC32 error counter |

Device Management

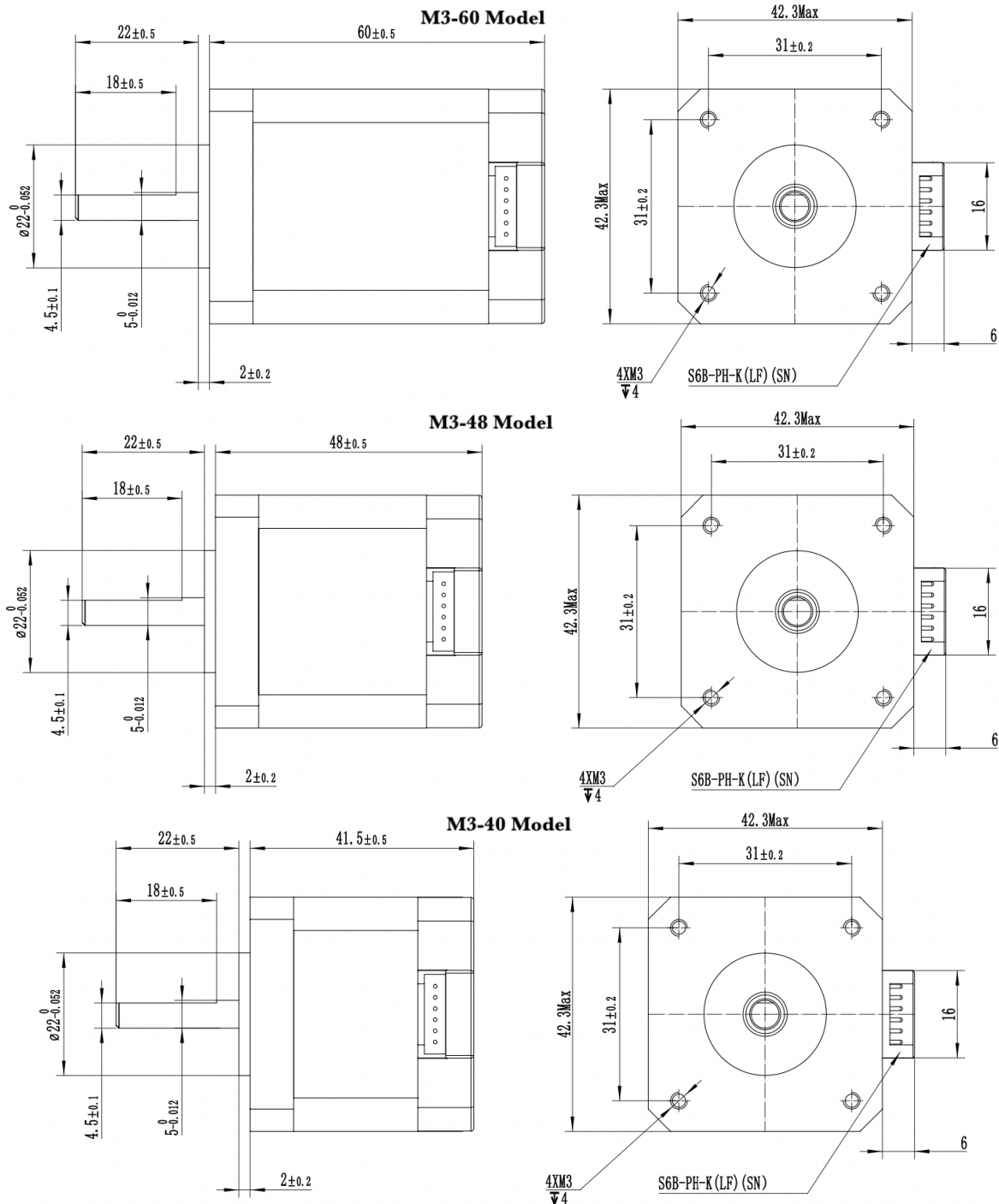
| Command | Description |
|-------------------------|---|
| Time sync | Sends the master time to the motor so that it can sync its own clock (do this 10 times per second). |
| Get product specs | Get the update frequency (reciprocal of the time step) |
| Detect devices | Detect all of the devices that are connected on the RS485 interface. Devices will identify themselves at a random time within one second. Chance of collision is possible but unlikely. You can repeat this if you suspect a collision (like if you have devices connected but they were not discovered within one to two seconds). |
| Set device alias | Sets device alias |
| Get product info | Get product information |
| Firmware upgrade | This command will upgrade the flash memory of the servo motor. Before issuing a firmware upgrade command, you must do some calculations as shown in the examples. |
| Get product description | Get the product description. |
| Get firmware version | Get the firmware version or the bootloader version depending on what mode we are in. This command also returns the status bits, where the least significant bit tells us if we are currently in the bootloader (=1) or the main firmware (=0) |
| Ping | Send a payload containing any data and the device will respond with the same data back |
| Vibrate | Cause the motor to start to vary the voltage quickly and therefore to vibrate (or stop). |
| Identify | Identify your motor by sending this command. The motor's green LED will flash rapidly for 3 seconds. |

Other

| Command | Description |
|--------------------------|--|
| Capture hall sensor data | Start sending hall sensor data (work in progress; don't send this command) |
| Read multipurpose buffer | Read whatever is in the multipurpose buffer (the buffer is used for data generated during calibration, going to closed loop mode, and when capturing hall sensor data) |

Mechanical Specifications

| Parameter | M17-60 | M17-48 | M17-40 |
|------------------|--------------|--------------|--------------|
| Dimensions (LxW) | 42.2x42.2 mm | 42.2x42.2 mm | 42.2x42.2 mm |
| Height | 59.8 mm | 48.6 mm | 41.6 mm |
| Shaft Length | 20.4 mm | 20.4 mm | 18.5 mm |
| Weight | 470g | 360g | 285g |
| Protection Class | IP20 | IP20 | IP20 |



Technical Specifications

| Parameter | M17-60 | M17-48 | M17-40 |
|-------------------|----------|----------|----------|
| Operating Voltage | 12-24V | 12-24V | 12-24V |
| Rated Torque | 0.65 N.m | 0.55 N.m | 0.42 N.m |
| Maximum Speed | 560 RPM | 560 RPM | 560 RPM |
| Maximum Current | 1.1A | 1.1A | 1.1A |
| Rated Power | 38W | 32W | 25W |

Operating Conditions

| Parameter | Specification |
|--------------------------|--------------------------------|
| Operating Temperature | 0C to +80C |
| Storage Temperature | -20C to +60C |
| Humidity Range | 20% to 80% RH (non-condensing) |
| Installation Environment | Indoor use only |

Python Library

Our Python library provides a comprehensive interface for controlling M17 Series Servomotors. The library handles all low-level communication protocols and unit conversions, making it easy to integrate servomotor control into your Python projects.

Installation

Install the servomotor library using pip:

```
pip3 install servomotor
```

Example: Basic Motor Control

The following example demonstrates how to connect to a servomotor, enable the mosfets, and perform a trapezoid move:

```

import servomotor
import time

# Connect to the servomotor
motor = servomotor.M3(port='/dev/ttyUSB0')

# Enable the mosfets (power on the motor)
motor.enable_mosfets()

# Wait for motor to be ready
time.sleep(0.5)

# Perform a trapezoid move
# Move 10000 steps with acceleration 1000, velocity 5000
motor.trapezoid_move(
    position=10000, # Target position in steps
    velocity=5000,  # Maximum velocity in steps/second
    acceleration=1000 # Acceleration in steps/second^2
)

# Wait for move to complete
motor.wait_for_move_complete()

# Move back to starting position
motor.trapezoid_move(
    position=0,
    velocity=5000,
    acceleration=1000
)

# Wait and disable mosfets
motor.wait_for_move_complete()
motor.disable_mosfets()

# Close connection
motor.close()

```

Key Features

- Support for all motor commands
- Automatic unit conversion for time, position, velocity, acceleration, current, voltage, and temperature
- Error handling and status monitoring
- Support for multiple motors on the same bus
- Get up and running in just a few lines of code

Documentation

For complete API documentation and more examples, visit:
https://github.com/tomrodinger/servomotor/tree/main/python_programs

Arduino Library

The Arduino library for M17 Series Servomotors provides an easy-to-use interface for controlling motors from Arduino boards. The library supports hardware serial communication and includes all essential motor control functions.

Installation

Install the library through the Arduino Library Manager:

- 1. Open Arduino IDE
- 2. Go to Sketch → Include Library → Manage Libraries
- 3. Search for "M17 Servomotor"
- 4. Click Install

Manual Installation

Alternatively, download the library from GitHub and install manually:
https://github.com/tomrodinger/servomotor/tree/main/arduino_library

Example: Basic Motor Control

This example shows how to initialize a motor, enable mosfets, and perform a trapezoid move:

```
#include <M17Servomotor.h>

// Create motor object on Serial1
M17Servomotor motor(&Serial1, 115200);

void setup() {
  // Initialize serial for debugging
  Serial.begin(115200);

  // Initialize motor communication
  motor.begin();

  // Wait for motor to be ready
  delay(1000);

  // Enable mosfets (power on)
  motor.enableMosfets();
  Serial.println("Motor enabled");

  // Wait for motor to stabilize
  delay(500);
}

void loop() {
  // Move forward 10000 steps
  Serial.println("Moving forward...");
  motor.trapezoidMove(
    10000, // position in steps
    5000,  // max velocity in steps/sec
    1000   // acceleration in steps/sec^2
  );

  // Wait for move to complete
  motor.waitForMoveComplete();
  delay(1000);

  // Move back to origin
  Serial.println("Moving back...");
  motor.trapezoidMove(0, 5000, 1000);

  // Wait for move to complete
  motor.waitForMoveComplete();
  delay(1000);
}
```

Hardware Connections

- Arduino TX → Motor RX (via level shifter if needed)
- Arduino RX → Motor TX (via level shifter if needed)
- Arduino GND → Motor GND
- Motor Power → 12-48V DC power supply

Key Features

- Simple, intuitive API for motor control
- Support for multiple motors on different serial ports
- Built-in error checking and timeout handling
- Compatible with Arduino Uno, Mega, Due, and other boards
- Minimal memory footprint

Additional Resources

For more examples and API reference, see the library documentation at:
<https://github.com/tomrodinger/servomotor/wiki/Arduino-Library>

Company Information

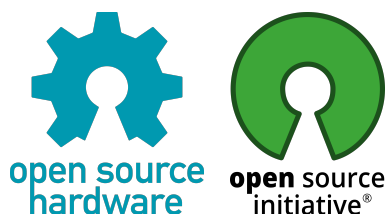
Green Eco Technology (Shenzhen) Company Limited Room C301E, 3F, Block CD, Tianjing Building Shatou Street, Futian District Shenzhen City, Guangdong Province, China

Open Source

We believe in making the world better through technology. All software, firmware, and PCB design files are available here:



<https://github.com/tomrodinger/servomotor>



Datasheet Version: 1.5 Release Date: Sep. 5, 2025

(c) 2024 All specifications subject to change without notice.
For more information and technical support, please contact our sales team.