Servomotor Arduino API Documentation

Version 1.5

Generated: 2025-09-19

Latest Firmware Versions

At the time of generating this API reference, the latest released firmware versions for the servomotors are:

• **Model M17:** servomotor_M17_fw0.13.0.0_scc3_hw1.3.firmware

Table of Contents

Getting Started

This example demonstrates a trapezoid move with the servomotor:

```
// Minimal Arduino example: Trapezoid move using built■in unit conversions
// Goal: spin the motor exactly 1 rotation in 1 second, then stop.
// Sequence:
//  enable MOSFETs -> trapezoidMove(1.0 rotations, 1.0 seconds) -> wait 1.1s -> disable MOSFETs.
//
// Notes:
// - This uses the library's unit conversion (no raw counts/timesteps).
// - Configure Serial1 pins for your board (ESP32 example pins below).
// - Motor is created AFTER Serial1.begin(...) so hardware UART pins are set first.

#include <Servomotor.h>

#define ALIAS 'X'                   // Device alias
#define BAUD 230400                 // RS485 UART baud rate
#define DISPLACEMENT_ROTATIONS 1.0f // 1 rotation
#define DURATION_SECONDS 1.0f       // 1 second
#define TOLERANCE_PERCENT 10        // +10% wait margin because the motor's clock is not
                                    //  perfectly accurate
#define WAIT_MS ((unsigned long)(DURATION_SECONDS * 1000.0f * (100 + TOLERANCE_PERCENT) / 100))

// Example RS485 pins for ESP32 DevKit (change as needed for your board)
#if defined(ESP32)
#define RS485_TXD 4             // TX pin to RS485 transceiver
#define RS485_RXD 5             // RX pin from RS485 transceiver
#endif

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

  // Create the motor; serial port opens on first instantiation.
#if defined(ESP32)
  Servomotor motor(ALIAS, Serial1, RS485_RXD, RS485_TXD);
#else
  Servomotor motor(ALIAS, Serial1);
#endif

  // Use units: rotations for position, seconds for time
  motor.setPositionUnit(PositionUnit::SHAFT_ROTATIONS);
  motor.setTimeUnit(TimeUnit::SECONDS);

  motor.enableMosfets();
  motor.trapezoidMove(DISPLACEMENT_ROTATIONS, DURATION_SECONDS);
  delay(WAIT_MS);
  motor.disableMosfets();
}

void loop() {
}
```

Data Types

This section describes the various data types used in the Servomotor API commands.

Integer Data Types

Type	Size (bytes)	Range	Description
i16	2	-32,768 to 32,767	16-bit signed integer
i24	3	-8,388,608 to 8,388,607	24-bit signed integer
i32	4	-2,147,483,648 to 2,147,483,647	32-bit signed integer
i48	6	-549,755,813,888 to 549,755,813,887	48-bit signed integer
i64	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	64-bit signed integer
i8	1	-128 to 127	8-bit signed integer
u16	2	0 to 65,535	16-bit unsigned integer
u24	3	0 to 16,777,215	24-bit unsigned integer
u32	4	0 to 4,294,967,295	32-bit unsigned integer
u48	6	0 to 1,099,511,627,775	48-bit unsigned integer
u64	8	0 to 18,446,744,073,709,551,615	64-bit unsigned integer
u8	1	0 to 255	8-bit unsigned integer

Special Data Types

Type	Size (bytes)	Description
buf10	10	10 byte long buffer containing any binary data
crc32	4	32-bit CRC
firmware_page	2058	This is the data to upgrade one page of flash memory. Contents includes the product model code (8 bytes), firmware compatibility code (1 byte), page number (1 byte), and the page data itself (2048 bytes).
general_data	Variable	This is some data. You will need to look elsewhere at some documentation or into the source code to find out what this data is.

list_2d	Variable	A two dimensional list in a Python style format, for example: [[1, 2], [3, 4]]
string8	8	8 byte long string with null termination if it is shorter than 8 bytes
string_null_term	Variable	This is a string with a variable length and must be null terminated
success_response	Variable	Indicates that the command was received successfully and is being executed. The next command can be immediately transmitted without causing a command overflow situation.
u24_version_number	3	3 byte version number. the order is patch, minor, major
u32_version_number	4	4 byte version number. the order is development number, patch, minor, major
u64_unique_id	8	The unique ID of the device (8-bytes long)
u8_alias	1	This can hold an ASCII character where the value is represented as an ASCII character if it is in the range 33 to 126, otherwise it is represented as a number from 0 to 255
unknown_data	Variable	This is an unknown data type (work in progress; will be corrected and documented later)

Command Reference

Basic Control

Disable MOSFETs

Disables the MOSFETS (note that MOSFETs are disabled after initial power on).

Example:

```
// Disable MOSFETs
motor.disableMosfets();
```

Enable MOSFETs

Enables the MOSFETS.

Example:

```
// Enable MOSFETs
motor.enableMosfets();
```

Reset time

Resets the absolute time to zero (call this first before issuing any movement commands)

Example:

```
// Reset time
motor.resetTime();
```

Emergency stop

Emergency stop (stop all movement, disable MOSFETS, clear the queue)

Example:

```
// Emergency stop
motor.emergencyStop();
```

Zero position

Make the current position the position zero (origin)

Example:

```
// Zero position
motor.zeroPosition();
```

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.

Example:

```
// System reset
motor.systemReset();
```

Configuration

Set maximum velocity

Sets maximum velocity (this is not used at this time)

Parameters:
• *maximumVelocity*: u32: Maximum velocity.

Example:

```
// Set maximum velocity
uint32_t maximumVelocity = 1000;

motor.setMaximumVelocity(maximumVelocity);
```

Set maximum acceleration

Sets max acceleration

Parameters:
• *maximumAcceleration*: u32: The maximum acceleration.

Example:

```
// Set maximum acceleration
uint32_t maximumAcceleration = 1000;

motor.setMaximumAcceleration(maximumAcceleration);
```

Start calibration

Starts a calibration, which will determine the average values of the hall sensors and will determine if they are working correctly

Example:

```
// Start calibration
motor.startCalibration();
```

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.

Parameters:

• *motorCurrent*: u16: The motor current. The units are some arbitrary units and not amps. A value of 150 or 200 is suitable.
• *regenerationCurrent*: u16: The motor regeneration current (while it is braking). This parameter is currently not used for anything.

Example:

```
// Set maximum motor current
uint16_t motorCurrent = 100;
uint16_t regenerationCurrent = 100;

motor.setMaximumMotorCurrent(motorCurrent, regenerationCurrent);
```

Set safety limits

Set safety limits (to prevent motion from exceeding set bounds)

Parameters:

• *lowerLimit*: i64: The lower limit in microsteps.
• *upperLimit*: i64: The upper limit in microsteps.

Example:

```
// Set safety limits
float lowerLimit = 0;
float upperLimit = 0;

motor.setSafetyLimits(lowerLimit, upperLimit);
```

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.

Parameters:

• *testMode*: u8: The test mode to use or trigger

Example:

```
// Test mode
uint8_t testMode = 1;

motor.testMode(testMode);
```

Set PID constants

Set PID constants for the control loop that will try to maintain the motion trajectory.

Parameters:
• *kP*: u32: The proportional term constant (P)
• *kI*: u32: The integral term constant (I)
• *kD*: u32: The differential term constant (D)

Example:

```
// Set PID constants
uint32_t kP = 1000;
uint32_t kI = 1000;
uint32_t kD = 1000;

motor.setPidConstants(kP, kI, kD);
```

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.

Parameters:
• *maxAllowablePositionDeviation*: i64: The new maximum allowable position deviation setting

Example:

```
// Set max allowable position deviation
float maxAllowablePositionDeviation = 0;

motor.setMaxAllowablePositionDeviation(maxAllowablePositionDeviation);
```

CRC32 control

Enable or disable CRC32 checking for commands

Parameters:
• *enableCrc32*: u8: Control value (1 to enable, 0 to disable CRC32 checking)

Example:

```
// CRC32 control
uint8_t enableCrc32 = 1;

motor.crc32Control(enableCrc32);
```

Device Management

Time sync

Sends the master time to the motor so that it can sync its own clock (do this 10 times per second).

Parameters:
• *masterTime*: u48: The motor absolute time that the motor should sync to (in microseconds).

Example:

```
// Time sync
float masterTime = 0;

timeSyncResponse response = motor.timeSync(masterTime);
```

Get product specs

Get the update frequency (reciprocal of the time step)

Example:

```
// Get product specs
getProductSpecsResponse response = motor.getProductSpecs();
```

Detect devices

Detect all of the devices that are connected on the RS485 interface. Devices will identify themselves at a random time within one seconde. 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).

Example:

```
// Detect devices
detectDevicesResponse response = motor.detectDevices();
```

Set device alias

Sets device alias

Parameters:
• *alias*: u8_alias: The alias (which is a one byte ID) ranging from 0 to 251. It cannot be 252 to 254 because those are reserved. You can set it to 255, which will remove the alias.

Example:

```
// Set device alias
uint8_t alias = 1;

motor.setDeviceAlias(alias);
```

Get product info

Get product information

Example:

```
// Get product info
getProductInfoResponse response = motor.getProductInfo();
```

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.

Parameters:
• *firmwarePage*: firmware_page: The data to upgrade one page of flash memory. Contents includes the product model code (8 bytes), firmware compatibility code (1 byte), page number (1 byte), and the page data itself (2048 bytes).

Example:

```
// Firmware upgrade
float firmwarePage = 0;

motor.firmwareUpgrade(firmwarePage);
```

Get product description

Get the product description.

Example:

```
// Get product description
getProductDescriptionResponse response = motor.getProductDescription();
```

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 significan bit teels us if we are currently in the bootloader (=1) or the main firmware (=0)

Example:

```
// Get firmware version
getFirmwareVersionResponse response = motor.getFirmwareVersion();
```

Ping

Send a payload containing any data and the device will respond with the same data back

Parameters:

• *pingData*: buf10: Any binary data payload to send to the device.

Example:

```
// Ping
float pingData = 0;

pingResponse response = motor.ping(pingData);
```

Vibrate

Cause the motor to start to vary the voltage quickly and therefore to vibrate (or stop).

Parameters:

• *vibrationLevel*: u8: Vibration level (0 = turn off, 1 = turn on).

Example:

```
// Vibrate
uint8_t vibrationLevel = 1;

motor.vibrate(vibrationLevel);
```

Identify

Identify your motor by sending this command. The motor's green LED will flash rapidly for 3 seconds.

Example:

```
// Identify
motor.identify();
```

Motion Control

Trapezoid move

Move immediately to the given position using the currently set speed (the speed is set by a separate command)

Parameters:
• *displacement*: i32: The displacement to travel. Can be positive or negative.
• *duration*: u32: The time over which to do the move.

Example:

```
// Trapezoid move
int32_t displacement = 1000;
uint32_t duration = 1000;

motor.trapezoidMove(displacement, duration);
```

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.

Parameters:
• *position*: i32: New absolute position value.
• *duration*: u32: Time allowed for executing the move.

Example:

```
// Go to position
int32_t position = 1000;
uint32_t duration = 1000;

motor.goToPosition(position, duration);
```

Homing

Homing (or in other words, move until a crash and then stop immediately)

Parameters:
• *maxDistance*: i32: The maximum distance to move (if a crash does not occur). This can be positive or negative. the sign determines the direction of movement.
• *maxDuration*: u32: The maximum time to allow for homing. Make sure to give enough time for the motor to cover the maximum distance or the motor may move too fast or throw a fatal error.

Example:

```
// Homing
int32_t maxDistance = 1000;
uint32_t maxDuration = 1000;

motor.homing(maxDistance, maxDuration);
```

Go to closed loop

Go to closed loop position control mode

Example:

```
// Go to closed loop
motor.goToClosedLoop();
```

Move with acceleration

Rotates the motor with the specified acceleration

Parameters:
• *acceleration*: i32: The acceleration (the unit is microsteps per time step per time step * 2^24).
• *timeSteps*: u32: The number of time steps to apply this acceleration. Use command 18 to get the frequency of the time steps. After this many time steps, the acceleration will go to zero and velocity will be maintained.

Example:

```
// Move with acceleration
int32_t acceleration = 1000;
uint32_t timeSteps = 1000;

motor.moveWithAcceleration(acceleration, timeSteps);
```

Move with velocity

Rotates the motor with the specified velocity.

Parameters:
• *velocity*: i32: The velocity (the unit is microsteps per time step * 2^20).
• *duration*: u32: The time to maintain this velocity.

Example:

```
// Move with velocity
int32_t velocity = 1000;
uint32_t duration = 1000;

motor.moveWithVelocity(velocity, duration);
```

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.

Parameters:

• *moveCount*: u8: Specify how many moves are being communicated in this one shot.
• *moveTypes*: u32: Each bit specifies if the move is a (bit = 0) MOVE_WITH_ACCELERATION_COMMAND or a (bit = 1) MOVE_WITH_VELOCITY_COMMAND.
• *moveList*: list_2d: A 2D list in Python format (list of lists). Each item in the list is of type [i32, u32] representing a series of move commands. Each move command specifies the acceleration to move at or the velocity to instantly change to (according to the bits above) and the number of time steps over which this command is to be executed. For example: '[[100, 30000], [-200, 60000]]'. There is a limit of 32 move commands that can be listed in this one multi-move command. Each of the moves takes up one queue spot, so make sure there is enough space in the queue to store all of the commands.

Example:

```
// Multimove
uint8_t moveCount = 1;
uint32_t moveTypes = 1000;
uint32_t moveList = 1000;

motor.multimove(moveCount, moveTypes, moveList);
```

Other

Capture hall sensor data

Start sending hall sensor data (work in progress; don't send this command)

Parameters:
• *captureType*: u8: Indicates the type of data to capture. Currently 1 to 3 are valid.
• *nPointsToRead*: u32: Number of points to read back from the device
• *channelsToCaptureBitmask*: u8: Channels to capture bitmask. The first three bits are valid, which will turn on (0) or turn off (0) that hall sensor channel
• *timeStepsPerSample*: u16: Aquire a sample every this number of time steps. Time steps happen at the update frequency, which can be read with the Get product specs command
• *nSamplesToSum*: u16: Number of samples to sum together to make one point to transmit back
• *divisionFactor*: u16: Division factor to apply to the sum of the samples to scale it down before transmitting it so that it fits into the returned data type, which is a 16-bit number per each hall sensor

Example:

```
// Capture hall sensor data
uint8_t captureType = 1;
uint32_t nPointsToRead = 1000;
uint8_t channelsToCaptureBitmask = 1;
uint16_t timeStepsPerSample = 100;
uint16_t nSamplesToSum = 100;
uint16_t divisionFactor = 100;

captureHallSensorDataResponse response = motor.captureHallSensorData(captureType, nPointsToRead, channelsToCapt
```

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)

Example:

```
// Read multipurpose buffer
readMultipurposeBufferResponse response = motor.readMultipurposeBuffer();
```

Status & Monitoring

Get current time

Gets the current absolute time

Example:

```
// Get current time
getCurrentTimeResponse response = motor.getCurrentTime();
```

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)

Example:

```
// Get n queued items
getNQueuedItemsResponse response = motor.getNQueuedItems();
```

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)

Example:

```
// Get hall sensor position
getHallSensorPositionResponse response = motor.getHallSensorPosition();
```

Get status

Gets the status of the motor

Example:

```
// Get status
getStatusResponse response = motor.getStatus();
```

Control hall sensor statistics

Turn on or off the gathering of statistics for the hall sensors and reset the statistics

Parameters:
• *control*: u8: 0 = turn off statistics gathering, 1 = reset statistics and turn on gathering.

Example:

```
// Control hall sensor statistics
uint8_t control = 1;

motor.controlHallSensorStatistics(control);
```

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.

Example:

```
// Get hall sensor statistics
getHallSensorStatisticsResponse response = motor.getHallSensorStatistics();
```

Get position

Get the current desired position (which may differ a bit from the actual position as measured by the hall sensors)

Example:

```
// Get position
getPositionResponse response = motor.getPosition();
```

Get comprehensive position

Get the desired motor position, hall sensor position, and external encoder position all in one shot

Example:

```
// Get comprehensive position
getComprehensivePositionResponse response = motor.getComprehensivePosition();
```

Get supply voltage

Get the measured voltage of the power supply.

Example:

```
// Get supply voltage
getSupplyVoltageResponse response = motor.getSupplyVoltage();
```

Get max PID error

Get the minimum and maximum error value ovserved in the PID control loop since the last read.

Example:

```
// Get max PID error
getMaxPidErrorResponse response = motor.getMaxPidError();
```

Get temperature

Get the measured temperature of the motor.

Example:

```
// Get temperature
getTemperatureResponse response = motor.getTemperature();
```

Get debug values

Get debug values including motor control parameters, profiler times, hall sensor data, and other diagnostic information.

Example:

```
// Get debug values
getDebugValuesResponse response = motor.getDebugValues();
```

Get communication statistics

Get and optionally reset the CRC32 error counter

Parameters:

• *resetCounter*: u8: Reset flag (1 to reset the counter after reading, 0 to just read)

Example:

```
// Get communication statistics
uint8_t resetCounter = 1;

getCommunicationStatisticsResponse response = motor.getCommunicationStatistics(resetCounter);
```