SMD4 User Manual

UHV Stepper Motor Drive

Introduction

SMD4_1_transparent.webp

The SMD4 stepper motor drive is a single-axis bipolar stepper motor drive, intended for use with AMLs range of vacuum-compatible stepper motors (VCSMs). It maximises motor performance while minimising temperature rise.

Powerful software is supplied with the SMD4 that enables you to easily control and configure multiple SMD4 units simultaneously, in a single, user-friendly graphical interface. For advanced users, the drive can also be controlled via scripting in JavaScript.

The drive features extensive communications options, including USB, RS232, RS485 and Ethernet. It also includes an innovative boost feature; this uses an internal circuit to increase the 48 V input up to 67 V for driving the motors. This allows the use of standard 48 V power supplies without compromising on motor performance.

Liability and warranty

AML assumes no liability and the warranty becomes null and void if the end-user or third parties:

- Disregard the information in this document
- Use the product in a non-conforming manner
- Make any kind of alterations (modifications, repair work, etc.) to the product
- Use the product with accessories not listed in the corresponding product documentation

We reserve the right to make technical changes without prior notice. The figures are non-committal.

Safety and warning notices

⚠ **WARNING!** All work described in this document may only be carried out by persons who have suitable technical training and the necessary experience or who have been instructed by the end-user of the product.

The safety of any system incorporating the instrument is the responsibility of the assembler of the system.

Use the instrument only as specified in this manual, otherwise the protection provided by the instrument might be impaired.

Ignoring this or subsequent safety information could lead to personal injury, or malfunction or permanent damage to the equipment.

Technical information

General

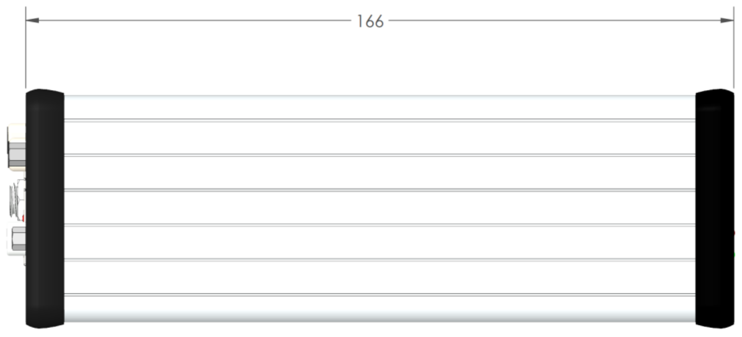
General	
Dimensions	166 mm x 106 mm x 56 mm (excluding connectors and feet)
Weight	0.5 kg
Protection class	IP 20
Temperature	Operation 10°C to 60°C, Storage -10°C to 85°C
Power supply	48 Vdc ± 5% power supply required. Power supply included.
Power consumption	48 W maximum
Safety compliance	EN 61010-1-2010
EMC compliance	Emissions EN61800-3:2018, EN55032 Class B, 3m (As 61800-3:2018, Table 17, Category C1, first environment) Immunities, EN55035, basic electromagnetic environment
Motor driver	
Type	2 phase bipolar stepper motor driver for 4-lead motors
Phase current	Up to 1 A RMS, adjustable in 30 mA steps
Source voltage	67 Vdc maximum 48 Vdc supply is boosted to 67 Vdc. Boost function can be disabled if required.
Resolution	Full, 8, 16, 32, 64, 128, 256 micro-stepping Stops on full step positions only, micro-stepping is used for control of resonance and smoother step transition.
Step frequency	1 Hz to 15 kHz
Protection	Short to ground and phase to phase
Motor temperature measurement	
Type	Selectable PT100 RTD or K-Type thermocouple
Range	-200°C to 240°C
Accuracy	±15 °C for thermocouple, ±5 % for RTD

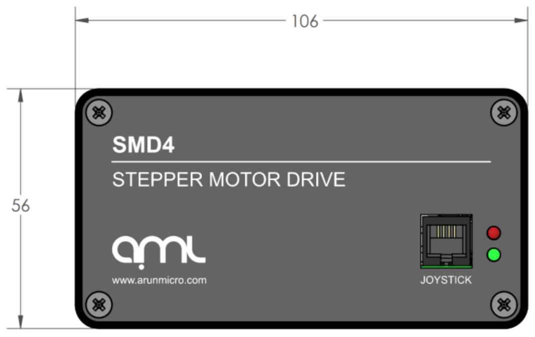
General	
Fault detection	RTD: Open and short-circuit Thermocouple: Open circuit only
Operating modes	
• Remote - Control and configure via USB, Ethernet or Serial • Step, Direction Enable (SDE) -For connection to an external motion controller or PLC • Joystick - Single-step and continuous movement triggered via a joystick (supplied separately) • Bake - Programmed cycle to heat the motor while stopped to drive off adsorbed gasses	
Control interfaces	
USB	USB 2.0 Full Speed via USB-C connector Virtual COM port and firmware update interface
Ethernet	10/100 Base-T, auto MDI-X, RJ45 8P8C connector Telnet (port 11312), Modbus TCP (port 502)
Serial communication	Selectable RS232 or RS485 mode (shared pins) Dual RJ45 8P8C connectors allow daisy chaining multiple devices in RS485 mode User selectable termination in RS485 mode 115200 default and maximum baud rate
Software	
Compatibility	Windows 10 or later
API	C# API is available
SDE (step, direction enable) interface	
Type	Optocoupled, common cathode
Levels	3.3 Vdc to 5 Vdc maximum Higher voltages require external current limiting resistor
Maximum frequency	2 MHz at 50% duty Maximum full-step rate limited to 7.8 kHz for micro-step resolution of 256.
Limits	
Quantity	2
Compatible switch types	Mechanical NO or NC (polarity selectable)
Protection	Withstands continuous short to 12 V maximum
Joystick	
Connection	Front panel mounted 4P4C jack with auto-detection of connection state
Input type	Active low, short to ground to activate function

General	
Miscellaneous	Open circuit voltage 3.3 V, source current < 3.5 mA

Mechanical

All dimensions are in millimetres.





Scope of delivery

Qty.	Item
1	SMD4
1	USB Type-A to USB Type-C lead
1	Power supply

Accessories

The following accessory items are available from AML.

Order Code	Item
SMD3JOY	Joystick (compatible with SMD4)
CAB-D15D9	SMD4 Cable, 3m, D-Sub 15 Male to D-Sub 9 Female
CAB-D15MLF	SMD4 Cable, 3m, D-Sub 15 Male to MLF18
CAB-3D15MLF	SMD4 Cable, 3m, 3X D-Sub 15 Male to MLF18

Installation

Before installation

WARNING: Read this manual carefully before installing and operating the SMD4. Observe the following safety instructions.

WARNING: All work described in this document may only be carried out by persons who have suitable technical training and the necessary experience or who have been instructed by the end-user of the product.

WARNING: Without proper training and necessary experience, damage to the equipment or personal injury might result.

DANGER: Danger of electric arcing! Never plug or unplug any connector while powered. Plugging or unplugging a motor while powered may damage or destroy the driver output stage.

Unpacking

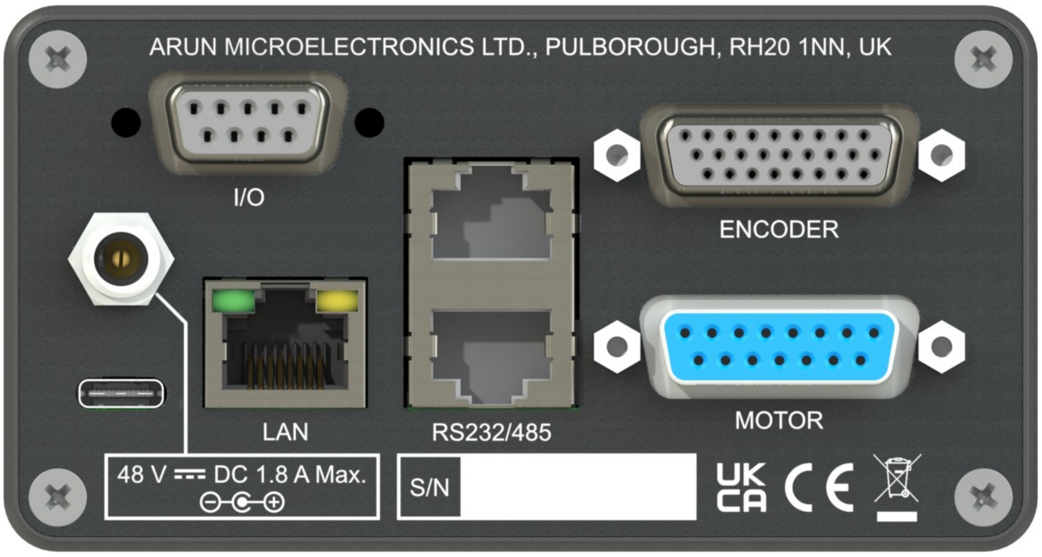
On receipt of the instrument remove all packing material and check that all items on the delivery note have been received. Report any damage or shortages to the company or distributor who supplied the instrument. The packing material has been specially designed to protect the instrument and should be retained for possible future use.

Mechanical installation

The SMD4 is a freestanding instrument. It does not require mounting. Forced air ventilation is not required. The ambient operating temperature range is 10 °C to 60 °C.

Connecting

Rear panel



Power

Connector: Barrel power jack 2.1 mm pin, 5.6 mm hole	
	48 Vdc input. Centre positive.

Power input for both internal logic circuits and the motor itself.

The power supply must:

- Meet the requirements set out in the technical information section of this document
- Provide reinforced or double insulation between mains and supply output

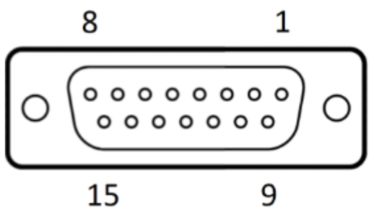
DANGER: Danger of electric arcing! Never plug or unplug the connector while powered.

CAUTION: In the event of reverse polarity, a short circuit will occur between GND and V+ through an internal power diode. An external fuse may be required.

The fuse should be sized:

- Greater than the current consumption of the SMD4 when operating the connected motor
- Less than the maximum current output of the power supply
- Considering the voltage of the supply

Motor

Connector: DA15-F D-Sub, 15 ways, female		
	1	Phase B2
	2	Phase B1
	3	Phase A2
	4	Phase A1
	5	Limit 1
	6	Limit 2
	7	Thermocouple negative
	8	Thermocouple positive
	9	GND
	10	
	11	
	12	

	13	RTD B2
	14	RTD B1
	15	RTD A

DANGER! Danger of electric arcing! Never plug or unplug the connector while powered! Plugging or unplugging motor while powered may damage or destroy the driver output stages.

Motor

Connection of the motor to the vacuum feedthrough, and vacuum feedthrough to the SMD4 is discussed in section Motor Wiring.

Limits

There are two limits inputs; they can be configured to stop the motor on one or both limits being triggered. A limit input is triggered by shorting it to 'GND', usually with a mechanical switch mounted on the mechanism that the motor is driving. Logic level signals, for example, from optical or hall effect sensors may also be used.
Input polarity can be reversed to accommodate normally open or normally closed switches.

Limit 1 applies when the motor position counter is incrementing, and limit 2 applies when the motor position counter is decrementing.

Limits inputs include a pullup resistor. See section Limits for details.

INFORMATION: Limits inputs are duplicated on the I/O connector for maximum user flexibility in arranging wiring for the system. Limit signals on both this and the I/O connector are electrically connected. Do not apply different electrical potentials between like-named limits inputs otherwise a short will occur between the two.

Thermocouple

The thermocouple lead for motors equipped with the standard K-Type thermocouple should be connected here. If using a motor equipped with an RTD, this connection may be left open. Be sure to select the correct sensor type, see section Temperature sensor selection.

INFORMATION: To provide greater convenience in wiring to the vacuum chamber, the thermocouple input is included on the motor connector, rather than a dedicated micro K-Type thermocouple connector. This comes at the cost of reduced accuracy due to the parasitic thermocouple junctions that exist within the connector. This is accounted for by the looser accuracy specification for the thermocouple input over the RTD one, and a generous tolerance in the motor over temperature threshold as further insurance. Nonetheless, accuracy can be improved by avoiding large temperature gradients across the SMD4; for example, avoid placing the rear panel of the product in direct line of hot exhaust from other equipment.

RTD

For motors equipped with an RTD instead of a thermocouple, make the RTD connection here. If the RTD is not required, leave the connections open.

The RTD input is compensated for cable length by the three-wire connection.

Custom cables

Custom motor cables must be built to the following specification to ensure continued compliance with EMC standards and correct function.

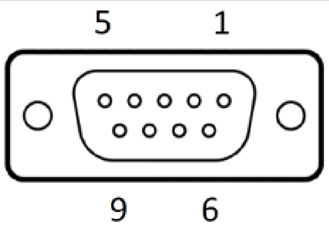
General requirements:

- Rated voltage >= 300 V rms, rated current > 1.5 A rms
- Construction; twisted pairs plus overall screen. Foil screen plus drain wire (of same or greater cross-sectional area as main cores) acceptable, foil plus braided screen better
- Screen must be connected via as short a wire as possible to a ground pin (pins 9-11) on the connector, using insulated wire
- Maximum cable length is limited by the resistance of the cores; total round-trip cable resistance per phase should be kept to less than a few ohms. Consult the cable manufacturer's data for these details. Excessive cable length causes a reduction in phase voltage at the motor compromising speed/torque characteristics. The RTD circuit is compensated against cable length.

Temperature sensors:

- Use one twisted pair for the thermocouple
- Use one twisted pair for RTD A and B1, and one lead from a second pair for B2

I/O

Connector: DE9-F D-Sub, 9 ways, female		
	1	GND
	2	Fault (Output, open collector)
	3	Limit 1/positive
	4	Enable
	5	Step
	6	Reset fault (Input, active low)
	7	Limit 2/negative
	8	SDE COM
	9	Direction

Fault output and fault reset

The SMD4 disables the motor under certain fault conditions, see sectionFaults. When this happens, the open collector ‘Fault’ output is set and may be used to signal to an external controller that the SMD4 is in a fault state.

Fault states are latching; once set the fault condition must be removed and the fault reset before normal operation may resume using either a remote interface command (see CLR) or pulling the ‘Reset fault’ signal to ‘GND’. This does not apply to the ‘EN’ (enable) input when in step direction mode, i.e. the enable input is not latching, and normal operation will resume immediately on restoring the enable input state. See section Faults for details.

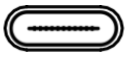
Limits

Duplicated from the motor connector.

Step, direction and enable

The step direction enable interface is an industry-standard interface allowing an external motion controller to generate stepping sequences, bypassing the SMD4’s internal motion controller. The inputs are galvanically isolated with three opto-isolators, and share a common connection, ‘SDE COM’. See section Step/Direction for details.

USB

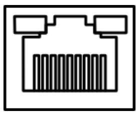
Connector: USB-C	
	

USB Type-C connection. The connection is reversible, and the plug may be inserted either way up.

The SMD4 appears as a virtual COM port when connected to the PC. No additional drivers are required. Configure and control the SMD4 using AML Device Control software, available as a free download from our website at https://arunmicro.com/documents/software/

Alternatively, use a terminal program, or your own application. AML supply a C# API, available on our website to help customers implement their own applications faster.

LAN

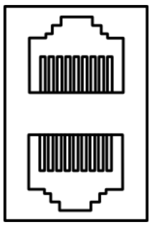
Connector: 8P8C RJ45 RJ45, 8 poles, 8 connections	
	

10/100M network connection with Auto MDI-X.

Configure and control the SMD4 using AML Device Control software, available as a free download from our website at https://arunmicro.com/documents/software/

Alternatively, use a terminal program, or your own application. AML supply a C# API, available on our website to help customers implement their own applications faster.

RS232/485

Connector: 8P8C RJ45 RJ45, 8 poles 8 connections		
	1	
	2	
	3	
	4	A+/Tx
	5	B-/Rx
	6	
	7	
	8	GND

RS232 and RS485 share pins. Before connecting to them, use an alternate interface (USB or LAN) to configure the desired mode of operation. Undefined behaviour will result if the SMD4 is connected to an RS485 network when in RS232 mode or vice-versa.

There are two identical connectors, allowing SMD4 devices to be daisy-chained together.

An optional termination resistance can be enabled if required. This would typically be enabled on the last device on the bus, to reduce reflections and maintain signal integrity. The usage of this feature should be evaluated in the final system.

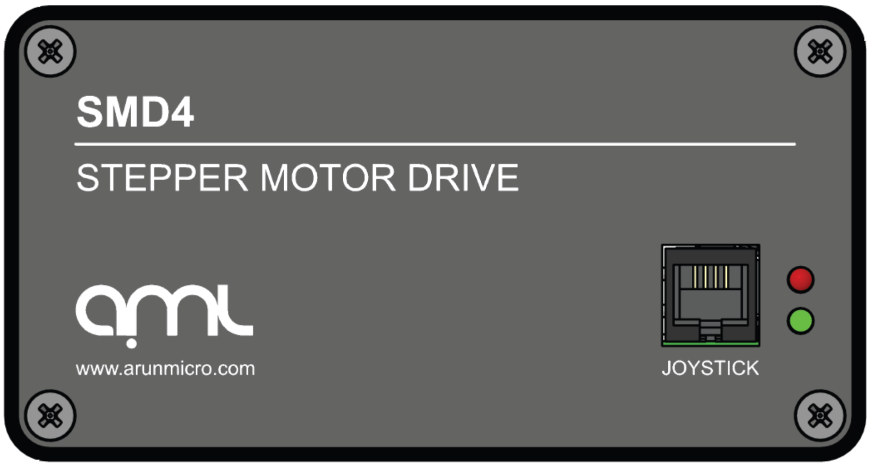
⚠ Configure the mode of operation (RS232 or RS485) before plugging a connector in. Do not make changes to the mode without first disconnecting both connectors. Undefined behaviour will result if the SMD4 is connected to an RS485 network when in RS232 mode or vice-versa.

Bussing

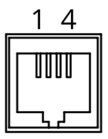
The dual connectors allow multiple SMD4s (or other devices) to be bussed together. There are several important considerations to be aware of when doing so:

- RS232 is not well suited to multi-drop; use RS485 for this purpose if at all possible. If using RS232, devices can receive and process commands, but will not respond and data cannot be returned to the host.
- Ensure all devices on the bus are in the the same mode (RS232 or RS485) and the same serial configuration (baud rate, etc) matches between devices.
- When using text protocol, begin all commands with the '@' addressing prefix. This places the SMD4 into addressing mode which brings into force alternate communication rules that allow the SMD4 to function properly on a bus with other devices. This is discussed in the Addressing section.
- Do not use text protocol commands without the addressing prefix. This will cause multiple devices to respond at once and the resulting bus contention may result in undefined behaviour.

Front panel



Joystick

	1	GND
	2	CW
	3	CCW
	4	DETECT

For connection of a two-button joystick allowing basic motor control, for example, during commissioning. AML supply the SMD3 Joystick, part number ‘SMD3JOY’, which is compatible with the SMD4, for this purpose. The SMD4 can be configured to automatically switch to joystick mode on connection of the joystick. See AUTOJS.

If designing your own joystick or device to connect to this port:

- Inputs have internal pull-ups
- Activate the function by shorting ‘CW’, ‘CCW’ or ‘DETECT’ to pin 1, ‘GND’
- ‘DETECT’ is used to signal to the SMD4 that the joystick is connected and trigger automatic switch to joystick mode (if configured). If this functionality is not required, leave the pin unconnected.

Logic level signals may also be used; 12 V max.

Status indicators

The fault indicator flashes or remains lit if the SMD4 is in a fault state (see sectionFaults for fault indications). When a fault is present, motor operation is disabled.

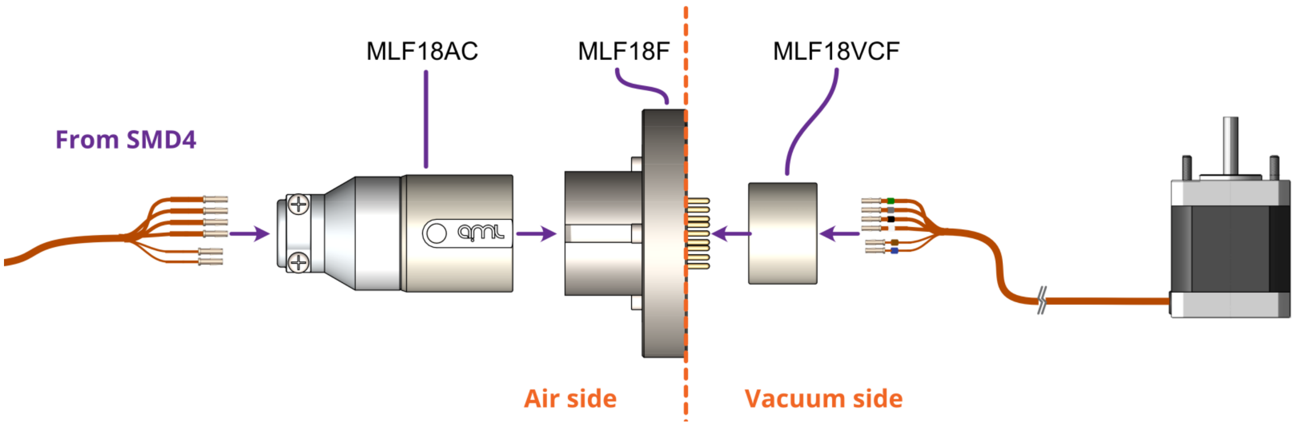
Motor wiring

Overview

Connecting motors inside a vacuum chamber to the SMD4 comprises two tasks:

- Wiring the motor to a vacuum feedthrough installed in the chamber wall.
- Wiring the vacuum feedthrough to the SMD4.

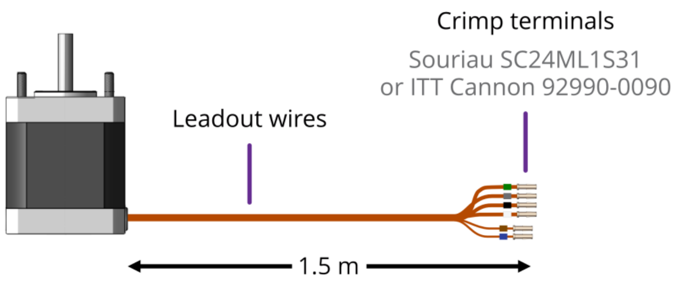
AML supply vacuum feedthroughs, ready-made cabling, and components allowing custom cables to be easily manufactured. A typical setup is shown below and used for illustration throughout this section.



INFORMATION: Verify that the motor is working correctly before sealing the vacuum chamber. Rectifying mistakes afterwards is inconvenient.

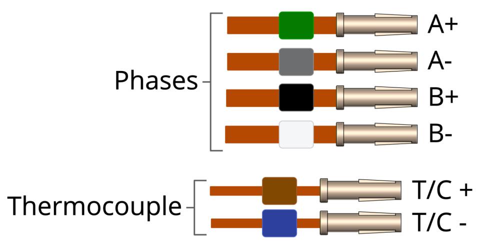
Lead identification

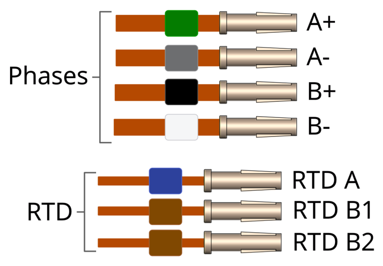
The motor leadout wires are self-coloured polyimide film-wrapped, silver-plated OFHC solid copper and each is fitted with a 1.5 mm crimp socket terminal. They are supplied fitted with UHV compatible coloured glass beads for identification. The phase leadout wires are much thicker than the thermocouple leadouts. The leadout wires of each phase should be twisted together.



Motors equiped with a ***Thermocouple:***

Motors equiped with an ***RTD:***





If the identification beads have been removed, the wires can be identified using an inexpensive multimeter, and a magnet. The multimeter must be capable of measuring resistance with a resolution of about 1 ohm.

Thermocouple Leadouts

The thermocouple wires are much thinner than the phase leads, and there are two of them. If three wires are present, the motor has an RTD installed, see below for details. The thermocouple is insulated from the rest of the motor.

The two leads are of different material; one is made from Alumel, which is weakly magnetic, and the other Chromel, which is not. Use a magnet to find the Alumel wire, then connect as shown below.

Lead	Connected to terminal marked	
Alumel	Alumel, N, - (minus) or coloured blue	🟦
Chromel	Chromel, P, + (plus) or coloured brown	🟫

RTD Leadouts

As per the thermocouple leads, but three instead of two leads. These must be identified by resistance; one pair of wires are connected at the motor end. These will measure a few ohms depending on cable length and are the ‘B1’ and ‘B2’ connections, which are interchangeable. The remaining wire is the ‘A’ connection and should measure around 100 ohms to either ‘B1’ or ‘B2’.

Lead	Connected to terminal marked	
A	A, or coloured blue	🟦
B1	Chromel,B1, or coloured brown	🟫
B2	B2, or coloured brown	🟫

Phase leadouts

These are the four thicker leadouts. Identify the two motor phases by their resistance, which will be in the range of 3 to 15 ohms, depending on the motor type. There is no electrical connection between the two phases, to the thermocouple/RTD or the case of the motor. Most of the resistance is in the windings of the motor and is virtually unaffected by shortening of the leads. Connect each phase to the appropriate drive terminals. The resistance of the wires from the feedthrough to the drive must be less than a few ohms.

Note regarding reversal of rotation

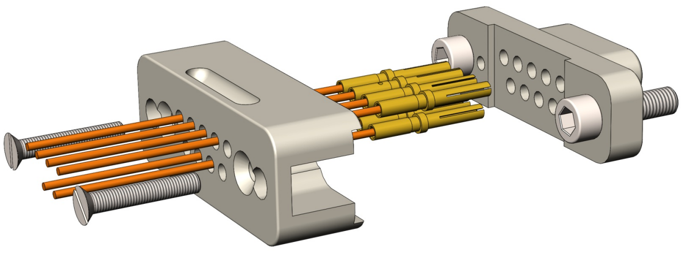
Upon completion of wiring, there is a 50 % probability that the direction of rotation will be reversed from the desired or conventional sense. To rectify this, exchange the connections to one of the phases. For example, locate the Phase A + and Phase A – connections, and swap them around. This can be done on air or vacuum side while the chamber is still open.

Wiring motor to a vacuum feedthrough

AML motors are commonly connected via an MLF18 feedthrough or a standard D-Sub feedthrough.

9-Way D-Sub

The VC9D-40CF 9-way D-Sub male feedthrough is suitable for one motor fitted with either a thermocouple or 3-wire RTD. The standard crimp terminals supplied with AML motor leadout wires should be removed and replaced with a VC9DF PEEK D-Sub female connector and crimp terminals. An optional VC9DB cable strain relief is also available.

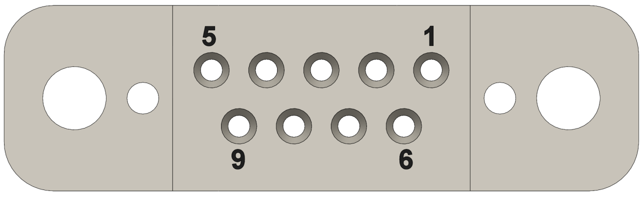


Motor wires pinout for the VC9DF

The illustration below shows the view into the non-mating side of the connector, into which the motor leads should be inserted, as shown below. Pass the wires through the backshell before crimping.

Connection	Colour	Pin
Phase A1	Green	4
Phase A2	Grey	3
Phase B1	Black	2
Phase B2	White	1
Thermocouple +	Brown	8
Thermocouple -	Blue	7
RTD A	Blue	5
RTD B1	Brown	6
RTD B2	Brown	9

Pin Insertion Side

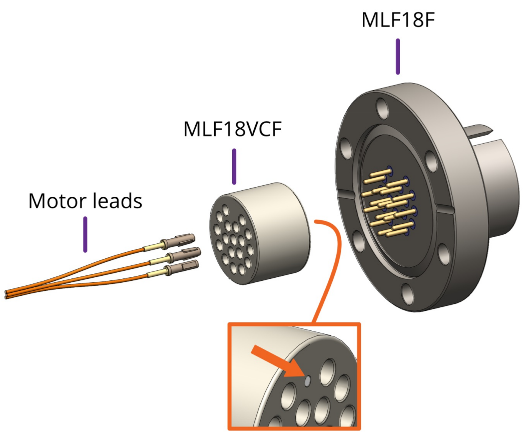


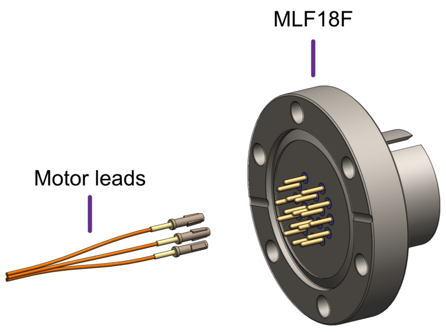
18-Way MLF18

The MLF18F feedthrough has 18 x 1.5 mm gold-plated feedthrough pins and is suitable for up to three motors fitted with thermocouples or up to two motors fitted with 3-wire RTDs. An internal bakeable connector, MLF18VCF, is available into which the crimp terminals on the motor leads are inserted. This significantly reduces the risk of short-circuits and makes the installation more convenient.

Using the MLF18F feedthrough and MLF18VCF vacuum side connector:

Alternatively, plug crimps directly onto the feedthrough pins of the MLF18F:



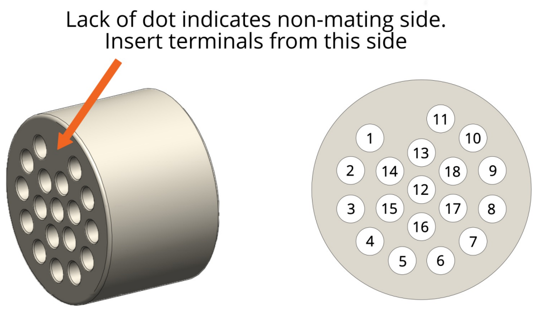


Mating side identified by dot. Motor lead terminals should be inserted in the other side.

Standard pinout for the MLF18VCF

The illustration below shows the view into the non-mating side of the connector, into which the motor leads should be inserted, as shown below.

Connection	Colour	Motor 1	Motor 2
Phase A1	Green	1	7
Phase A2	Grey	3	9
Phase B1	Black	2	8
Phase B2	White	4	10
Thermocouple +	Brown	5	11
Thermocouple -	Blue	6	12
RTD A	Blue	13	16
RTD B1	Brown	14	17
RTD B2	Brown	15	18

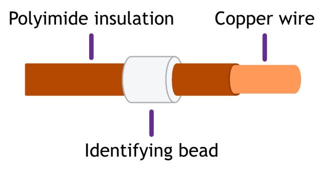

Lack of dot indicates non-mating side.
Insert terminals from this side
11
1
10
13
2
14
18
9
12
3
15
17
8
16
4
7
5
6

Using other feedthroughs
AML stepper motors can be ordered with either a K-Type thermocouple, or 3-wire PT100 RTD. The former requires 6 pins, and the latter 7 pins.

When using motors installed with a thermocouple, it is not necessary to use a thermocouple vacuum feedthrough or extension wires, as the error introduced by incompatible feedthrough material is usually less than 5 °C and the temperature measurement is not required to be very precise.

Preparation of motor leadouts for connection to other feedthroughs

If making custom terminations for the motor leads, the installed crimps must be removed, and the wire ends stripped of insulation. Standard motors are fitted with Polyimide film-wrapped leads (illustrated below), and radiation-hard motors are fitted with polyimide-enamelled leads.



Polyimide is strong, flexible and abrasion-resistant and therefore difficult to strip. The simplest method of stripping polyimide film is to cut a ring with a sharp knife and withdraw the cylinder of insulation over the end of the wire.
Be careful not to mark the conductor surface with the knife. Strip the enamelled radiation-hard leads by scraping with a sharp knife. Either type of lead may be stripped with a suitable high-speed rotary stripper. Do not use a thermal stripper.

Wiring between drive and vacuum feedthrough

AML supply three standard SMD4 cables. These are:

CAB-D15D9: 3 metres long. D-sub 15-way male connector for connecting to the SMD4. D-sub 9-way female connector for connecting to a feedthrough, such as AML's VF9D-40CF.

CAB-D15MLF: 3 metres long. D-sub 15-way male connector for connecting to the SMD4. MLF18AC connector for connecting to AML's MLF18F feedthrough.

CAB-3D15MLF: 3 metres long. 3x D-sub 15-way male connectors for connecting up to 3 separate SMD4s. MLF18AC connector for connecting to AML's MLF18F feedthrough.

Alternatively, AML supply multiple air-side connectors which can be used to make custom leads to mate with the MLF18F or D-sub electrical feedthroughs. These are supplied with a kit that includes the crimps as well as instructions for their use.

Leads between the MLF18AC and SMD4 should be assembled according to the following guidance for safe, reliable operation and continued compliance with EMC standards.

Cable requirements

- Quantity of cores as required; (one motor requires 6 cores when fitted with a thermocouple, or 7 if fitted with an RTD). The cable must be screened. A foil screen plus drain wire is acceptable; a foil plus braid screen is better.
- The screen must be connected via as short a wire as possible to pin 1, ‘GND’ of the motor connector, using insulated wire.
- Rated voltage >= 300 V rms
- Rated current > 1.5 A rms
- Cable cores must be twisted together in pairs, using one pair per phase, one pair for the thermocouple, and a group of three for the RTD. This reduces radiated emissions from the cable and improves immunity of the RTD and thermocouple signals to the motor.
- Maximum cable length is limited by round trip resistance, which should be less than a few ohms. Review cable manufacturer's data to obtain this figure.

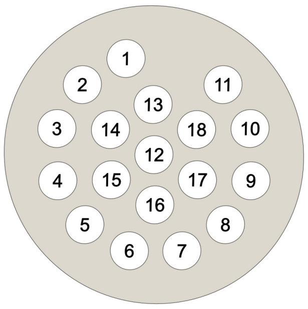
Wiring up to the MLF18AC airside connector

The MLF18AC is supplied with comprehensive instructions detailing correct usage of the connector. The pinout to match with the standard MLF18F + MLF18VCF pinning described in section Motor wiring is shown below. Note that the illustration shows the MLF18AC looking into the non-mating side of the connector, i.e. the side into which crimps are inserted.

Connection	Colour	Motor 1	Motor 2
Phase A1	Green	1	7
Phase A2	Grey	3	9

Looking into the non-mating face of the MLF18AC, into which crimps are inserted

Phase B1	Black	2	8
Phase B2	White	4	10
Thermocouple +	Brown	5	11
Thermocouple -	Blue	6	12
RTD A	Blue	13	16
RTD B1	Brown	14	17
RTD B2	Brown	15	18



Operation

Getting started

The quickest way to get started with the SMD4 having completed wiring up according to the previous chapter is to install the SMD4 software on your PC (see Software section), power on the SMD4 and connect it with the USB lead to the PC. The SMD4 software provides an intuitive and easy way to configure and evaluate the features of the SMD4 described in the remainder of this section.

This section discusses the various operational modes and configuration options available. The SMD4 software provides easy access to these functions, as well as help text describing each. If communicating with the SMD4 directly, using a terminal program or your own software, see section 7 which lists the commands available.

Operating modes

The product has several operating modes which define how motor movement is commanded. Configuration can be performed at any time and in any mode using one of the remote interfaces (USB, LAN or Serial).

The modes are:

- Remote - Control of motor via remote interface
- Joystick - Control of motor via a joystick
- Step/Direction - Control of motor via industry-standard step, direction enable interface
- Bake - Control of phase current to heat motor (while keeping it stationary) to drive off adsorbed gasses

Mode can be changed only when the motor is not performing any movement. This can be verified by checking the standby flag, which is returned in a status register by the SMD4 on every communication. The firmware prevents mode change when the standby flag is not set.

Remote

Command motor movements via the remote interface. Comprehensive control of all aspects of motor movement. Other functions are executed from remote mode, for example, homing, in which the motor moves until a limit switch is hit, serving as a reference for future movements.

The easiest way to use remote mode is with the supplied AML Device Control software, which allows one or more SMD4 units to be combined into a system and controlled individually or as a group. This makes it easy to apply the same configuration to multiple devices, for example.

Alternatively, the SMD4 may be controlled via a simple terminal application or your own software. A C# API is available to speed up development of your own applications. The remote interface is described in section remote interfaces.

Homing function

Homing drives the motor to the positive or negative limit switch. The motor first moves toward the limit switch using the existing movement profile. On the limit switch being triggered, the step frequency is halved, and the motor reversed until the limit switch is not triggered. Finally, the motor moves toward the limit switch at a step frequency of 30 Hz until the limit switch is triggered.

INFORMATION: Limitations of limits
Limit switches are not latching, i.e. as soon as a limit input becomes not triggered, for example if the mechanism is able to first actuate a limit switch and then continue moving past it until the limit switch is no longer actuated, then the SMD4 will be unaware of this and will continue to drive the motor if commanded.

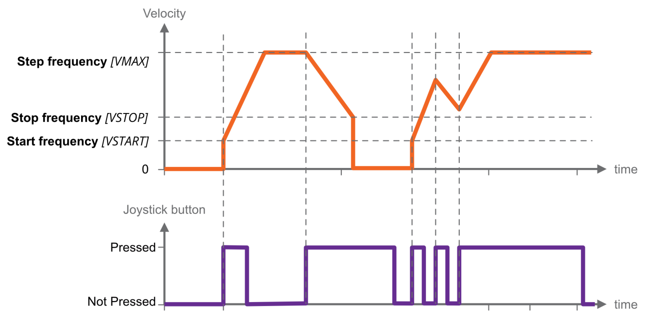
Limits switches and cams are normally arranged such that the limit switch is triggered from the desired point up to and including the point at which the mechanical limit of the mechanism is encountered.

Joystick

Basic motor movements may be commanded via a two-button joystick connected to front panel connector. AML supply the SMD4 Joystick, part number 'SMD4JOY' for this purpose. Optionally, on connection, the SMD4 automatically switches to joystick mode, and reverts to the previous mode on removal of the joystick. This behaviour can be disabled if required.

There are two joystick modes; both operate using velocity mode (see sectionVelocity and Positioning Mode for details) in which a profile, including acceleration, deceleration and target frequency are programmed, then motor movement is triggered by the joystick.

Continuous



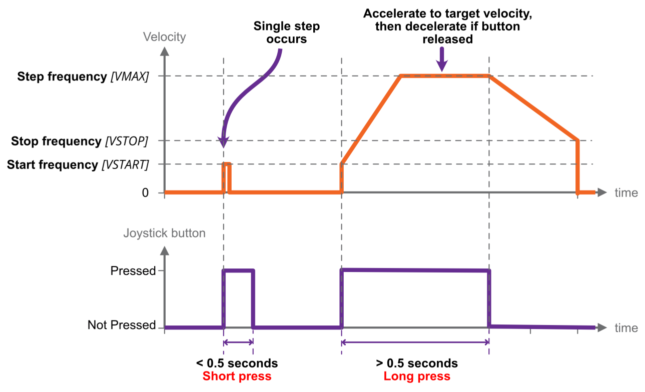
Motor accelerates toward target frequency on joystick key down. Continuing to hold the key down has no further effect.

On releasing and pressing the same key again, the motor decelerates toward a stop.

On pressing the alternate direction key, motor first decelerates to a stop before accelerating toward target frequency in the other direction.

If the motor has not yet come to a stop, and the same key is pressed again, the motor will once more accelerate towards target frequency, as illustrated left.

Single step



A short button press (< 0.5 s) causes a single step in the commanded direction. This is useful for precise positioning.

A long press (> 0.5 s) triggers acceleration toward the target frequency, while the button continues to be pressed.

Releasing the button causes the motor to decelerate toward a stop.

If the button is pressed while the motor is still decelerating, the motor once more accelerates toward target frequency for as long as the button is held.

Step/Direction

Motor movement is controlled by externally supplied step and direction signals. There are two sub modes, normal and triggered.

Normal mode

The SMD4 can be configured to step on the rising or rising and falling edges, which halves the step clock rate.

The external enable fault is non-latching when in step direction mode; once the external enable state is restored, or the external enable setting is changed to false, normal operation will resume immediately without the need to clear it.

Step input

	Both	Rising only
Rising	Step	Step
Falling	Step	

Direction input

	Meaning
Low	Positive
High	Negative

Steps are generated according to the current resolution. For example, with the edge setting on rising only, and microstep resolution set to 128, each rising edge on the step input will generate a single 1/128th step.

A step interpolation option is available; when enabled, the step input behaves as it would with the current resolution, except that each step input is interpolated to 256 microsteps. This is done by evaluating the rate at which steps arrive and timing 256 microsteps within the step-to-step period. This gives all the benefits of microstepping at high resolution while minimising the input clock rate.

The relationship between step input, resolution and actual step frequency is given below:

Motor Step frequency [Hz] = (Step input [Hz] / Resolution)

INFORMATION: Stopping on fractional steps
There is no mechanism to prevent the motor from stopping on fractional steps as there is in all other modes. Stopping on fractional steps will result in the motor temperature rising much faster than it otherwise would and is generally not suitable for vacuum applications. Therefore, configure the external step generator to meet this criteria.

INFORMATION: Preparation before switching out of Step/Direction mode
When changing to another mode from Step/Direction mode, ensure that any movement being commanded via Step/Direction interface has completed, and that the motor is at a full step position before switching.

Triggered mode

This mode works the same as joystick continuous mode, except that the positive and negative inputs that would normally be supplied via the joystick input are instead generated from the step and direction inputs:

Step input

	Meaning
Rising	Triggers start / stop
Falling	No action

Direction input

	Meaning
Low	Positive
High	Negative

In the case of Step/Direction mode, it is the responsibility of the external controller to perform any final activities, such as coming to a stop, before changing the mode.

Bake

Heats the motor by energising both phases and holding the motor stationary, regulating the current to achieve a set point temperature. Used to drive off adsorbed moisture in the motor.

Before engaging bake mode, set the target bake temperature. When in bake mode, the green status indicator will flash briefly at intervals as a reminder that this mode is active.

General concepts

User interface

In general, all control and configuration of the SMD4 is performed via the remote interface. The following functions and indications are available locally on the SMD4:

- Basic status information, via front panel green and red indicators. Green signifies power on and normal operation, red a fault. See section Faults. The green indicator also blinks briefly when the operating mode is changed.
- Joystick control – plug a joystick into the front panel joystick connection, and basic movements may be performed according to the current configuration. See section Joystick.
- Step/Direction interface; if in this mode, the motor may be controlled via signals supplied on the I/O port. See section Step / Direction.
- Fault output and fault reset input on the I/O connector – An open collector fault output is set when a fault

occurs. The fault state can be reset by pulling the fault reset pin to the ‘GND’ pin of the I/O connector. See section Faults.

Persistence of settings

All changes made to the configuration via the remote interface are volatile (i.e. not retained on power cycling) unless the store command is executed before powering off. The AML Device Control software warns you of this when closing the application, but if writing a custom application to control the SMD4, your application must handle this if settings are to be persisted.

The SMD4 will always load the last stored settings on power on, or if the store command has not been previously used, defaults are loaded as per section Command Reference. If settings become corrupted, for example, the write endurance of the memory in which the settings are stored is exceeded, the SMD4 loads defaults as identified above, and a fault indication is given, see section Faults.

INFORMATION: Write endurance
The memory in which settings are stored has an endurance of about 1 Million write cycles. Only use the store command when necessary, for example, take care that your application does not perform multiple redundant store commands.

Motor current

Three values may be set for motor current; acceleration current, run current and hold current.

	Applies when
Acceleration	Motor is running but not at target frequency, i.e. during acceleration and deceleration. This allows you to set a higher current during acceleration (to overcome inertia of a large load, for example) and revert to a lower current once the load is moving, thus reducing motor power dissipation and extending run time. If you do not wish to use this feature, simply set acceleration current to equal run current.
Run	Motor has reached target frequency.
Hold	Motor is stopped and is only necessary where the motor detent torque is not enough to prevent undesirable movement of the load. The cost of using hold current is increased motor temperature under vacuum. Therefore, where possible, mechanisms should be designed to be statically balanced, and the hold current should be set to 0.

When the motor starts moving, acceleration current is applied immediately. When the motor stops after the deceleration, two additional states must be traversed before the acceleration current is reduced to hold current, first, a configurable delay during which acceleration current continues to be applied (called ‘standstill’ state), followed by a configurable delay during which acceleration current is reduced to hold current (called ‘going to standby’ state).

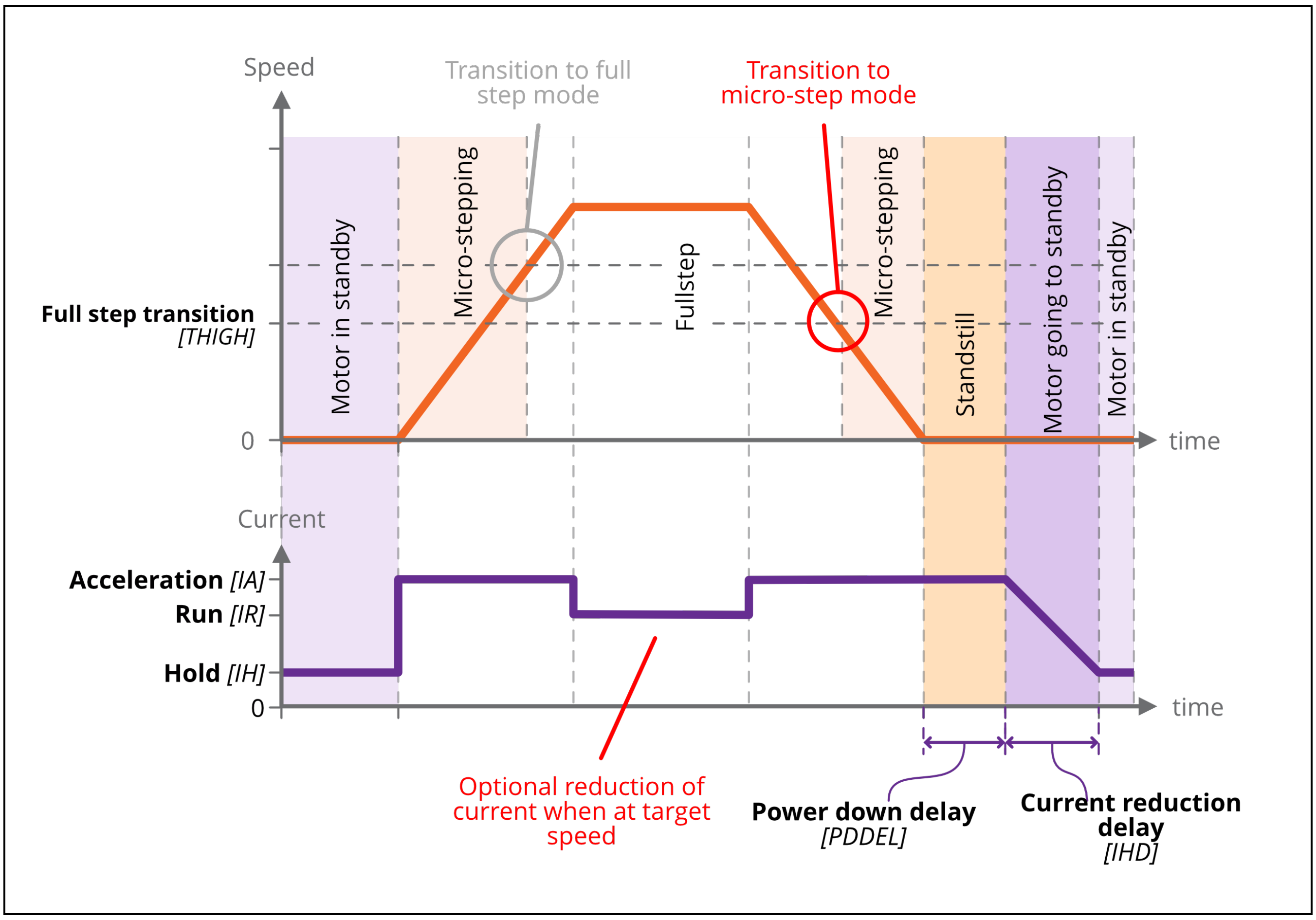
Run current must be set equal to or smaller than acceleration current. This is enforced by the SMD4; if a change to run current makes it greater than the acceleration current, the acceleration current is automatically adjusted to be equal to run current.

Standstill
Period after motor has stopped during which acceleration current is still applied. Adjustable between about 0 and 5.57 seconds using the ‘Power down delay’ setting [PDDEL]. Set to the minimum value suitable for your application to minimise heat generated.

Going to standby

Period during which acceleration current is gradually reduced to hold current. This smooth transition avoids a motor jerk on power down. Motor current is not continuously adjustable, instead being one of 31 discrete values from 0 to 1.044 A rms. Therefore, the current ramps down in steps. The step size may be set between 0 (instant power down) and 327 ms using the ‘Current reduction delay’ setting [IHD]. Set to the minimum value suitable for your application to minimise heat generated.

The name of each setting in the following illustrations matches that used in the software. The command mnemonic, for use if programming the SMD4 via the remote interface (see section remote interfaces), is given in square brackets.



Speed
Transition to full step mode
Transition to micro-step mode
Full step transition *[THIGH]*
Motor in standby
Micro-stepping
Fullstep
Micro-stepping
Standstill
Motor going to standby
Motor in standby
0
time
Current
Acceleration *[IA]*
Run *[IR]*
Hold *[IH]*
0
time
Optional reduction of current when at target speed
Power down delay *[PDDEL]*
Current reduction delay *[IHD]*

Figure 1 - Motor current and speed

Microstepping

Microstepping is applicable at low step frequencies (typically < 500 Hz) and helps reduce motor resonances resulting in smoother operation. In non-vacuum applications, it is also used to achieve increased positioning resolution, however, it requires energising both motor phases continuously even when the motor is stopped to maintain position, resulting in unacceptable levels of temperature rise in the motor in vacuum applications. Instead, mechanisms are designed to achieve the required positioning resolution with appropriate gearing.

Microstepping is not helpful at higher step rates, therefore, the SMD4 automatically switches between microstepping at low speeds and full step at high speeds. The transition point from full step to microstep is configurable, as illustrated in Figure 1. Hysteresis is applied to this value resulting in the transition in the opposite direction (from microstep to full step) being at a slightly higher frequency, as illustrated above. Note that you cannot explicitly set the transition to full step point, only the transition from full step to microstep. The other transition is calculated automatically.

The resolution to use during microstepping is configurable, via the microstep resolution setting [RES]. Choices are 8, 16, 32, 64, 128 and 256. In all modes except for step/direction, the motor is only stopped in full-step positions. Microstepping is used exclusively for the purpose of smoothing the transition between steps.

The accuracy with which motor profile (acceleration, deceleration, etc.) settings may be made depends on the microstepping resolution; the maximum microstep resolution of 256 offers the greatest accuracy for these settings.

Freewheel mode

Freewheel mode refers to how the motor is configured when it is at standstill and zero hold current is set. There are three choices:

- Use **freewheel** for minimum holding torque, which allows the motor shaft to be moved freely.
- **Phases shorted** for maximum holding torque with zero power applied to the motor (and so no heat generated in the motor).
- **Normal** offers some minimal amount of holding torque as a result of the phases still being connected to the driver circuitry.

Velocity and Positioning mode

All modes except step/direction fundamentally use the hardware of the SMD4 in one of two ways:

- **Velocity mode.** Motor is accelerated to a target velocity in a specified direction (Positive or Negative), which may be maintained indefinitely. On stopping, the SMD4 decelerates according to the configured profile and stops in a full step position.
- **Positioning mode.** Motor is driven toward a chosen position, determined by an internal step counter and a relative or absolute step count position. The motor starts by accelerating towards the target velocity, then as the target position is approached begins to decelerate before coming to a stop at the target position. Position can be specified in full steps only.

Step/direction triggered velocity mode uses velocity mode internally and joystick mode uses a combination of velocity and positioning mode. In remote mode, velocity and positioning mode must be selected with the appropriate command.

Initiating movement

In remote mode, movement is started via a run command and stopped via stop command. Movement cannot begin spontaneously as a result of changing a setting, for example. Direction is specified as positive, which results in the position counter incrementing, or negative which results in the position counter decrementing.

In all other modes, motor movement is determined by an external stimulus, for example, a joystick button press in joystick mode, or an edge on the step input in step/direction mode.

Monitoring the motor

Motor temperature, speed and position may be queried using remote interface commands.

INFORMATION: Motor control is open loop, therefore quantities such as position and velocity are determined from internal counters in the SMD4. These cannot be relied upon in certain circumstances, such as if the motor is stalled, or misses steps due to improper configuration.

Enable input

An enable input is present as part of the SDE interface, but the enable signal may be used in any mode. The motor is enabled when high and disabled (all motor movement inhibited) when low.

The enable input is ‘gated’ by an external enable setting; when enabled, the behaviour described above applies. When disabled, the enable input is treated as if it was true, regardless of the actual state. This allows the user to decide whether the enable input is used or not.

When the enable input is not used, then the SMD4 is responsible for enabling the motor as required, consistent with any other requirement described in this document.

Enable input

	Setting enabled	Setting disabled
Low	Motor disabled	Motor enabled
High	Motor enabled	Motor enabled

Motor configuration

Temperature sensor selection

AML motors can be supplied with either a K-Type thermocouple or PT100 RTD temperature sensor. Ensure the sensor is connected to the thermocouple or RTD input on the motor connector, and make the appropriate selection. The temperature sensor select command allows selection between thermocouple and RTD.

INFORMATION: The motor is disabled if the temperature sensor is misconnected, faulty or the temperature measurement exceeds 190 °C in order to protect the motor from possible damage to the insulation material.

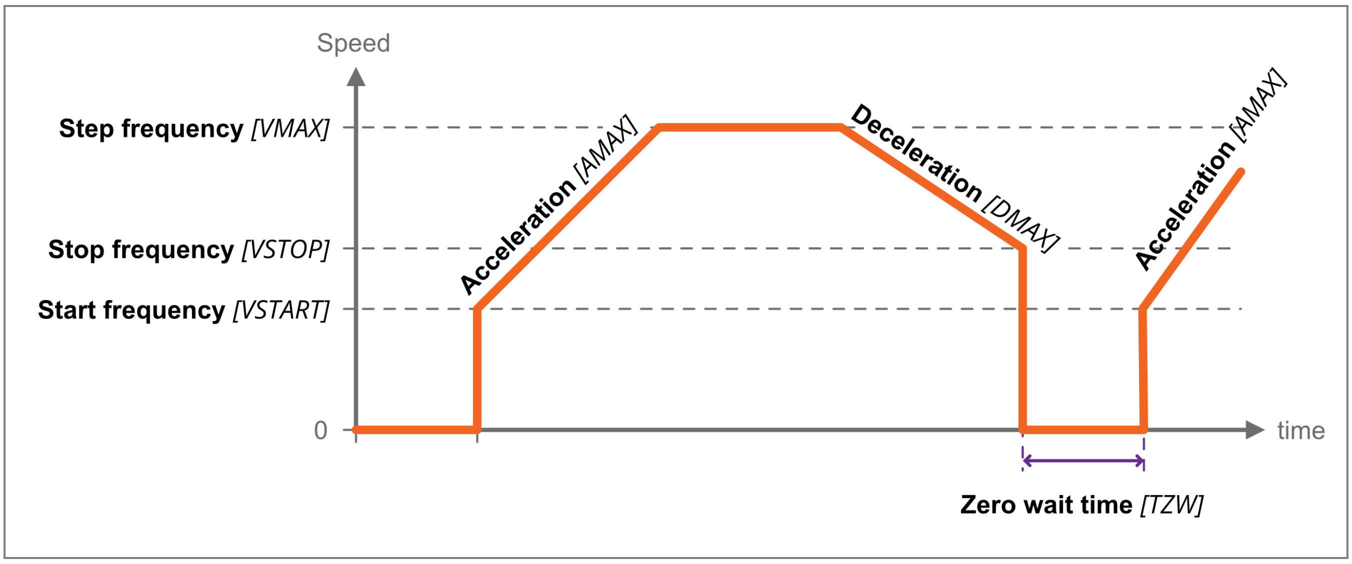
Check that the motor temperature sensor selection matches that of your motor.

When using a thermocouple, avoid significant temperature gradients across the thermocouple leads and connector on the SMD4.

Profile configuration

This section is concerned with configuring dynamic properties of motor movement. The profile settings apply to all modes except for step/direction mode. Tuning of these parameters is required to optimise motor performance in your application, and is necessary to engage positioning or velocity mode.

The name of each setting in the following illustrations matches that used in the software. The command mnemonic, for use if programming the SMD4 via the remote interface (see section remote interfaces), is given in square brackets.



Start and stop frequency

The start frequency is the initial step rate, and helps to allow the motor to overcome inertia and start moving smoothly; if start frequency were zero, the duration of the initial few steps might be long enough that the motor would overcome inertia on the first step, then effectively stop for a period of time, then have to overcome inertia once more for the second step, and so on, until the steps were frequent enough that the motor remains moving.

Stop frequency is the counterpart setting which determines the frequency for the last step. Both are specified in Hz, or steps per second.

Start frequency must be set equal to or smaller than stop frequency. This is enforced by the SMD4; if a change to the stop frequency makes it smaller than the start frequency, start frequency is automatically adjusted to be equal to stop frequency.

Start frequency and Stop frequency must be set equal to or smaller than Step frequency. The SMD4 will not force Start and Stop frequency to match Step frequency if either Start or Stop frequency are smaller than Step frequency.

Acceleration and deceleration

The SMD4 implements linear acceleration and deceleration ramps; velocity ramps up between the start frequency and the step frequency, and down linearly between the step frequency and stop frequency. Both are specified in Hz^{-1}, or steps per second per second.

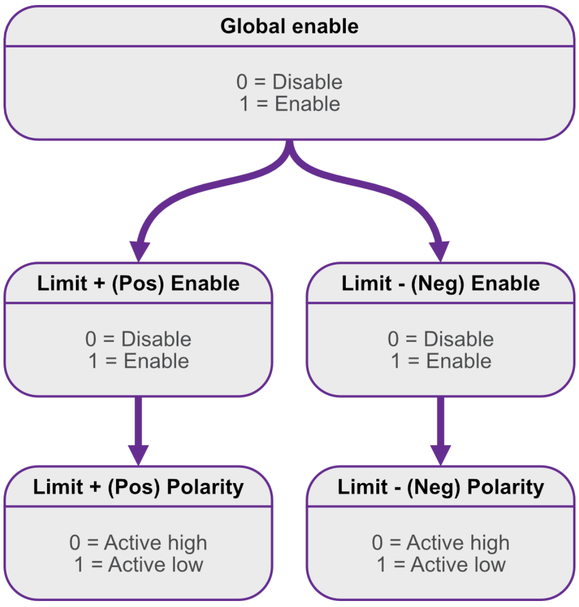
Changing direction

When using higher values for the start and stop frequency, a subsequent move in the opposite direction would result in a jerk equal to start frequency + stop frequency. The motor may not be able to follow this. The zero wait time setting can be used to introduce a short delay between the two and eliminate the jerk as illustrated above.

Limits

Mechanisms sometimes include limit switches, typically mechanical switches positioned to open or close when some start or end point of travel is reached for example. This is used as a trigger to stop the motor

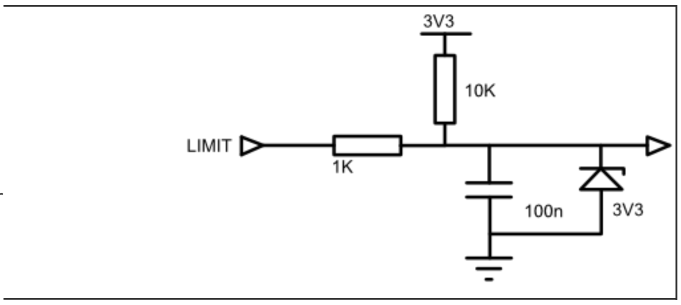
Configuration options allow limits to be individually or globally enabled or disabled, polarity to be set (i.e. trigger limit on switch open, called ‘Active low’ or switch closed, called ‘Active high’). The configuration options are illustrated below. AML Device control software additionally allows each limit to be given a custom name, for example ‘right switch’ to make it easier to identify within the software.



Limit activation summary in relation to direction

	Limit + (Pos) triggered	Limit − (Neg) triggered
Positive	Motor disabled	Motor enabled
Negative	Motor enabled	Motor disabled

Limits are intended for use with the mechanical switches typically found in vacuum mechanisms, but other types of switch, for example, hall effect or optical limit switches can be interfaced. The high and low logic thresholds are 1.5 V and 0.55 V respectively. Limit inputs can withstand up to 12 V maximum. The limits input circuit is outlined below for reference.

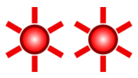


Faults

If a fault occurs, the motor is stopped and power to the motor is removed. The cause of the fault is indicated via the red front panel indicator, as shown below, and reflected in the error flags available via the remote interface.

Types of fault

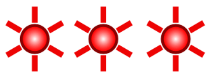
●	**Internal** A hardware or software malfunction inside the SMD4. For example, user settings have become corrupted and failed to load properly.
✹	**External enable criteria not satisfied** The external enable setting is ‘true’ and the SMD4 requires the external enable signal to be high to enable the motor. Either supply an enable signal, or if you do not wish to use the enable input, disable the external enable setting by setting it to ‘false’, which tells the SMD4 to ignore the state of the enable signal.



Motor temperature

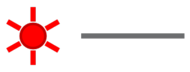
Motor temperature has exceeded 190 °C or a fault has been detected with the temperature sensor. Excessive temperature can damage the insulation on the motor windings, and the SMD4 does not allow the motor to be driven. The SMD4 shuts down the motor before this can happen to prevent possible damage to the motor. Wait for the motor to cool before attempting to run the motor again.

> **INFORMATION:** The motor is also disabled if the temperature sensor is misconnected or fault in order to protect the motor from possible damage to the insulation material.



Motor short

A motor short has been detected. Motor phase-to-phase and phase-to-ground shorts can be detected by the SMD4. Inspect the motor and wiring to resolve before attempting to run the motor again.



Limit hit

Indicator flashes briefly once a second. A limit has been triggered and has stopped the motor.

Clearing a fault

Faults may be cleared using the clear command, or by pulling the fault reset pin to the ‘GND’ pin the I/O connector.

The external enable fault is non-latching when in step direction mode; once the external enable state is restored, or the external enable setting is changed to false, normal operation will resume immediately without the need to clear it as described above.

Software

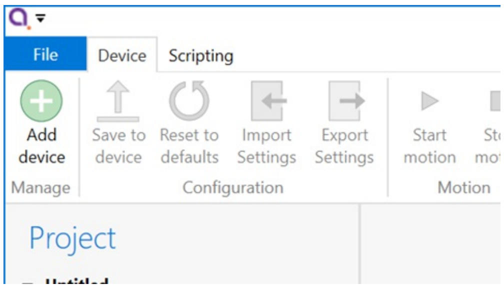
Installation and setup

The SMD4 is compatible with the AML Device Control software, which can be downloaded from the Software page on our website: https://arunmicro.com/documents/software/

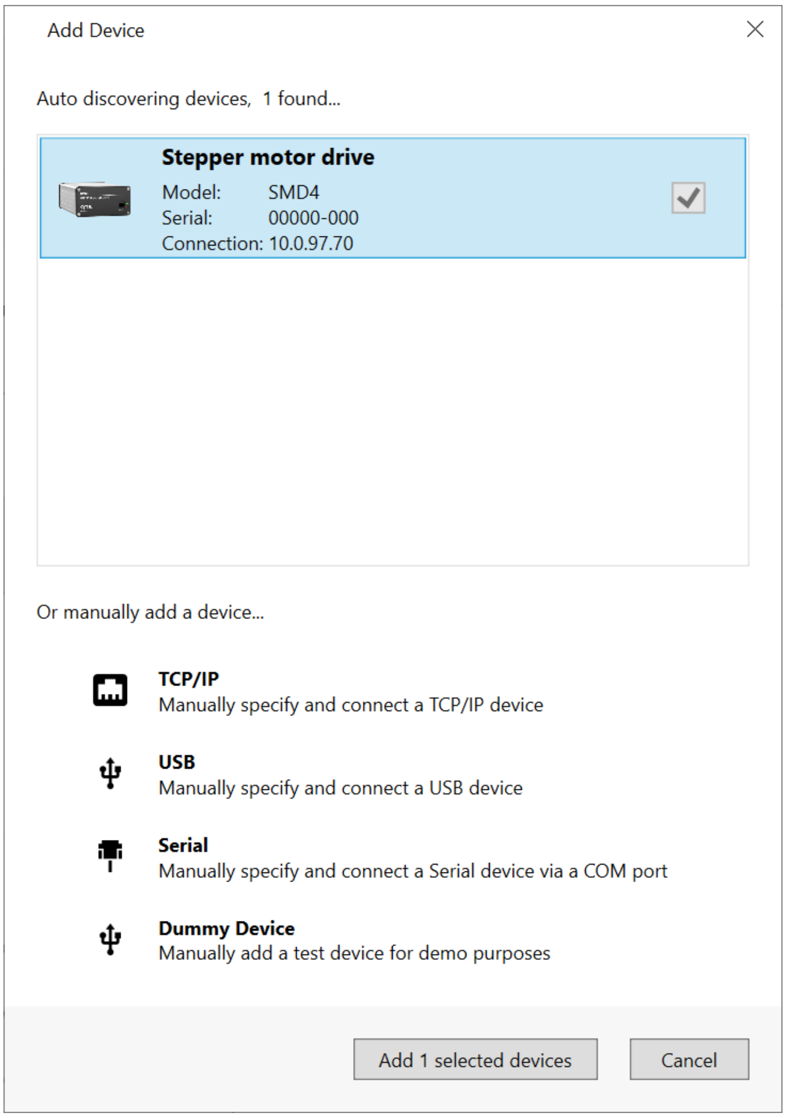
- Connect all SMD4 devices to your computer

  > If you intend to ultimately connect via LAN or Serial, start off by using the USB interface to perform basic setup on the LAN or serial interface first, then once satisfied connect on LAN or Serial as required.
  >
  > Multiple instances of the same physical SMD4, but on different interfaces are allowed. For example, you could plug in the SMD4 with a network cable and USB lead, add the device connected via USB, configure network settings, then add device again as a network device. This can be useful during commissioning in establishing a working setup.

- Start the AML Device Control software and click ‘Add device’ in the top left corner



USB connected SMD4 devices should automatically appear in the list. Select all devices that you wish to add and click “Add n selected devices”



Add Device

Auto discovering devices, 1 found...

Stepper motor drive
Model: SMD4
Serial: 00000-000
Connection: 10.0.97.70

Or manually add a device...

TCP/IP
Manually specify and connect a TCP/IP device

USB
Manually specify and connect a USB device

Serial
Manually specify and connect a Serial device via a COM port

Dummy Device
Manually add a test device for demo purposes

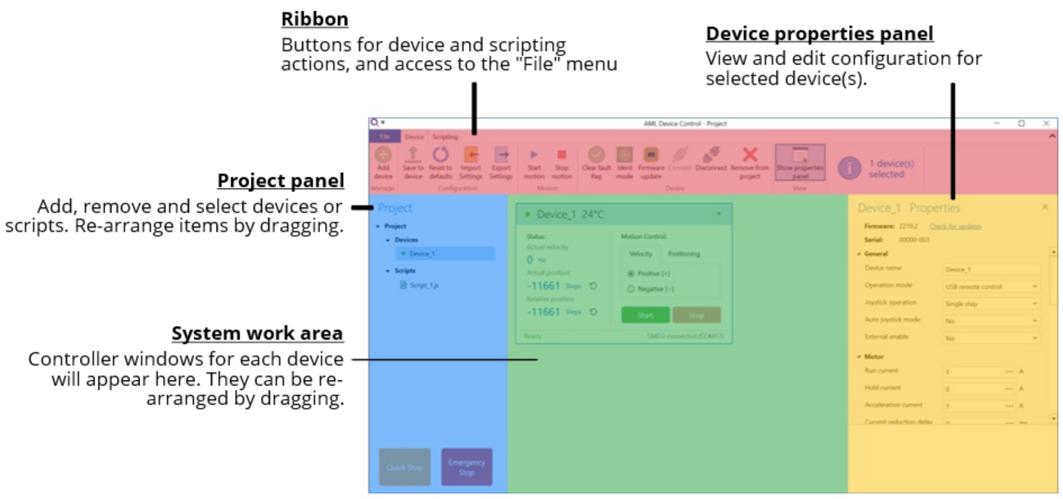
Add 1 selected devices | Cancel

The speed of auto detection varies by interface quantity of ports on your PC. Detection of USB and LAN devices is typically quick, whereas detection of devices connected via RS232 or RS485 can be considerably slower if a large number of COM ports are present on the PC.

The SMD4 network interface has an implementation of SSDP (Simple Service Discovery Protocol) which allows it to be discovered easily on a network, without knowing it's IP address.

Overview

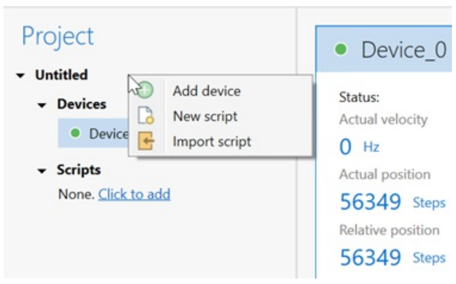
The default layout of the software is shown below.


Ribbon
Buttons for device and scripting actions, and access to the "File" menu
Device properties panel
View and edit configuration for selected device(s).
Project panel
Add, remove and select devices or scripts. Re-arrange items by dragging.
System work area
Controller windows for each device will appear here. They can be re-arranged by dragging.

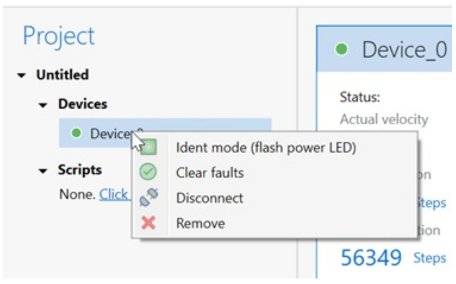
Project panel

Shows a list of the devices and scripts in the project. Currently selected devices are highlighted. Multiple devices can be selected by holding down CTRL and clicking. The device properties panel shows the properties for the selected device(s).

Right-clicking on an empty area within the project panel presents a context menu, with options to add new devices, new scripts or importing scripts.



The right-click context menu on each device provides access to functions such as clearing faults or placing the selected device into ident mode in which the green status indicator flashes.



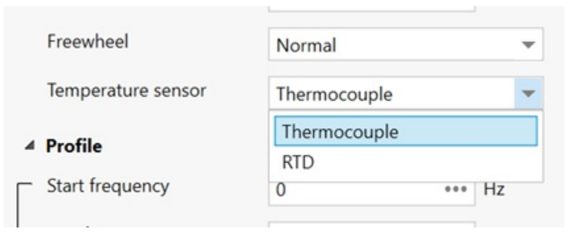
A status indicator next to each device shows the current status of each device according to these colours:

Colour	Description
🟢	Device connected and ready
🔵	Bake mode running, limit switch triggered or joystick connected
🔴	Device in a fault state
⚫	Device disconnected

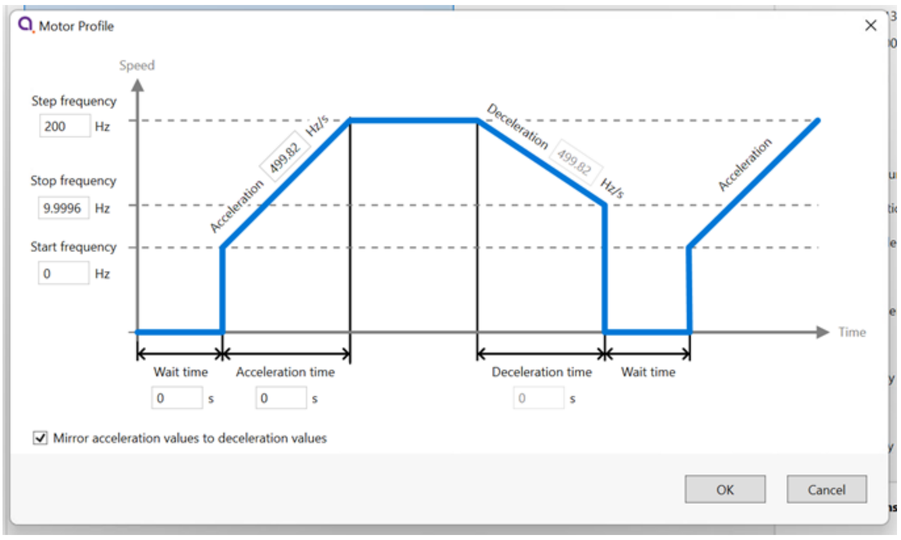
Device properties panel

Select one or more devices in the project panel; their properties are displayed and can be edited here. A blank is shown for properties which are different across the selected devices. As each property is selected, help text appears at the bottom of the panel describing the configuration option in more detail.

- Some properties allow selection of one of several choices, for example, temperature sensor selection:



- Others such as ‘Run current’ simply require a numeric value to be entered
- Others allow values to be entered directly, or the three dots button to the right to be clicked. This opens a window, allowing a more complex value to be input, for example, ‘Acceleration’ allows input in the native units or seconds:



System work area

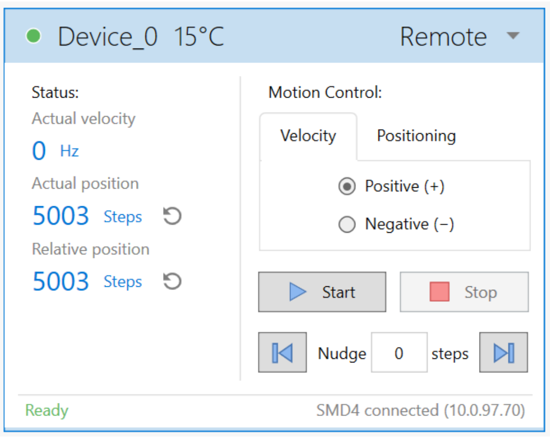
Controller windows for each device appear in this area. Windows can be arranged as desired and will automatically 'snap' to a grid making it easy to keep them neatly organised.

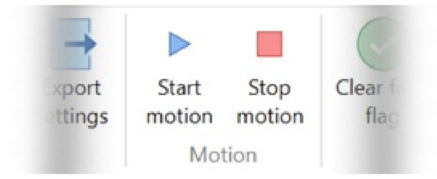
Controller window

Shows a status summary for the selected device, providing essential information such as actual velocity, actual (absolute) position, relative position and error status.

Absolute and relative position counters may be reset using the ↺ icons.

For controlling an SMD4, choose the type of motion and click start or stop. Multiple SMD4 devices can be controlled using the motion controls on the ribbon:

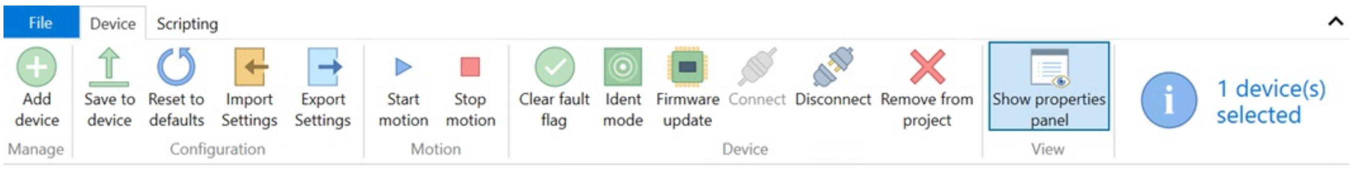




Be aware that the synchronisation between multiple SMD4s is not specified or guaranteed when controlling them in this way; delays within the computer, software, and data connection to the SMD4 mean that each SMD4 will start or stop its motion at a slightly different moment, therefore this option is not suitable for performing complex co-ordinated movements across multiple axes.

Ribbon

Contains buttons for all actions. Within the software, buttons can be hovered over for more information.



Saving projects

SMD4 configuration is maintained in two locations:

1. The SMD4 itself, with the use of the "Save to device" command. If the "Save to device" command is not used, settings will revert to their previous values on power cycling.
2. In the software project file.

The behaviour of the software in relation to this is as follows:

If the serial number of a connected SMD4 matches that of one in the project file that is open: The configuration given in the project file prevails, and the SMD4 configuration is synchronised to match that in the project file. Note that unless you use the save changes to hardware function, the original configuration of the SMD4 is not overwritten, and will be restored on power cycle or by using the load command to restore the configuration from flash.

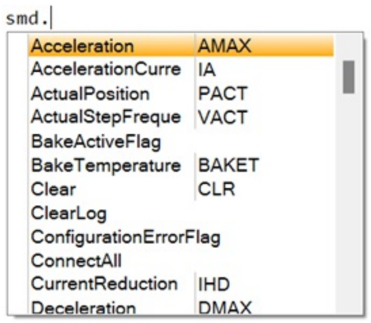
If the serial number of a connected SMD4 does not match any of those in the project file that is open: The SMD4 is considered a new device in the project, and the project file will be initialised from the configuration found in the SMD4 itself. After this point, the first behaviour outlined above applies.

INFORMATION: In the first case above, configuration items that require the SMD4 to be in standby will not be correctly synchronized if the SMD4 is not in standby when the software connects.

Scripting

The software includes an easy to use script editor, that allows for sequences to be programmed and executed on multiple connected SMD4 devices, as well as system level operations such as adding and removing SMD4 devices from the project.

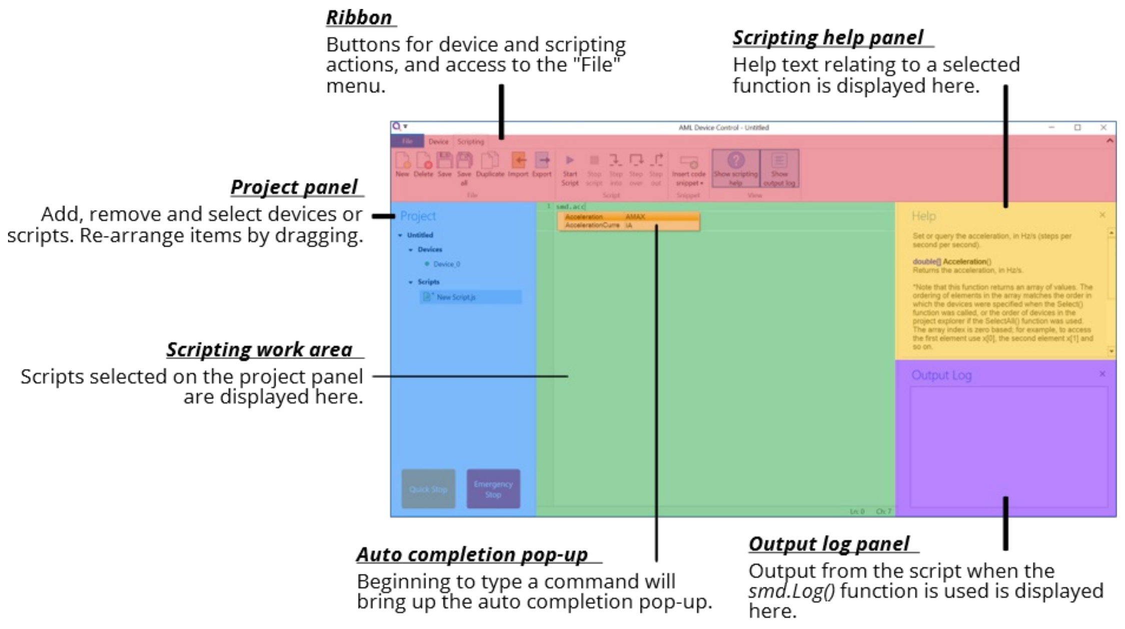
The scripting language used is JavaScript; this is powerful, easy to use and extensively documented. A global ‘smd’ object is made available from which you perform all interactions with the SMD4s. Type ‘smd.’ and an auto completion popup appears, showing all available commands, as well as help documentation for each. Press the enter key to select an option, then provide any arguments required.



```
smd.
```

Acceleration	AMAX
AccelerationCurre	IA
ActualPosition	PACT
ActualStepFreque	VACT
BakeActiveFlag	
BakeTemperature	BAKET
Clear	CLR
ClearLog	
ConfigurationErrorFlag	
ConnectAll	
CurrentReduction	IHD
Deceleration	DMAX

A brief description of each function/command is presented below the scripting section. Here is an overview of the scripting area:



Ribbon
Buttons for device and scripting actions, and access to the "File" menu.

Scripting help panel
Help text relating to a selected function is displayed here.

Project panel
Add, remove and select devices or scripts. Re-arrange items by dragging.

Scripting work area
Scripts selected on the project panel are displayed here.

Auto completion pop-up
Beginning to type a command will bring up the auto completion pop-up.

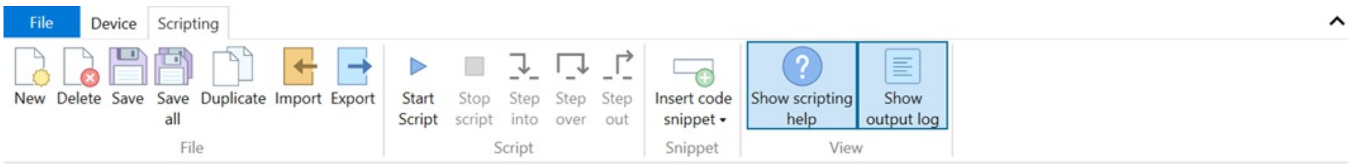
Output log panel
Output from the script when the *smd.Log()* function is used is displayed here.

The auto completion popup can be shown using the ‘Ctrl-K’ keyboard shortcut.

Information on the available SMD4 device specific commands can be found in sectionUSB of this manual. Serial command mnemonics will be auto completed by the script editor. The arguments of each scripting function are identical to those shown in section Command Reference, however, the format of querying and commanding is different. The example below shows how the SMD4 mode can be set and queried:

```
smd.Mode(2);                                    // Set the SMD4 mode to 2 (remote)
smd.Mode();                                     // Query state of mode
```

The ribbon contains scripting specific buttons.



Function specific to the SMD4 software

Function	Description
Add	bool **Add**(string **serial**) Add a new device to the project. Returns true if the device has been detected and added to the project. serial: Device serial number.
ClearLog	void **ClearLog**( ) Clear command line.
ConnectAll	void **ConnectAll**( ) Connect all devices in the project.
Delayms	void **Delayms**(int **value**) Delay in milliseconds. value: Minimum: 0 Maximum: 2^31 -1
DelaySeconds	void **DelaySeconds**(int **value**) Delay in seconds. value: Minimum: 0 Maximum: 2147483
DisconnectAll	void **DisconnectAll**( ) Disconnect all devices in the project.
Log	void **Log**(string **value**) Print value to the command line.
Name	bool **Name**(string **serial**, string **name**) Change name of the device. Returns true if the device has been found and name changed. serial: Device serial number. name: Device new name.
Remove	bool **Remove**(string **serial**) Remove a device from the project. Returns true if the device has been detected and removed from the project. serial: Device serial number.
RemoveAll	void **RemoveAll**( ) Remove all devices from the project.
Select	bool **Select**(string[ ] devices) Returns true if all the requested devices have been selected. devices: Name of the device(s).

SelectAll	void **SelectAll**( ) Select all devices.
SelectNone	void **SelectNone**( ) Deselect all devices.

Functions that query the SMD4 and that are not also available as a remote command are listed below.

Note that these all return an array rather than a single value, with each array element corresponding to the data from one SMD4. Use the array index syntax to access the desired element. This applies regardless of the number of devices.

For example, if there is only one SMD4 connected, use BakeActiveFlag()[0] to get the state of the bake active flag for that device.

The order of the array elements matches the order in which the SMD4 devices are selected. For example, suppose the “X-axis”, “Y-axis” and “Z-axis” named devices were selected with the command “smd.Select(“Z-axis”, ”X-axis”, ”Y-axis)”, to check the standby flag of the “Z-axis” device, use MotorStandbyFlag()[2]. Notice that array indices are 0 based.

Function	Description
BakeActiveFlag	bool[] **BakeActiveFlag**() Returns true if the bake mode is running.
ConfigurationErrorFlag	bool[] **ConfigurationErrorFlag**() Returns true if the motor configuration is corrupt.
EmergencyStopFlag	bool[] **EmergencyStopFlag**() Returns true if the motor is disabled by software.
ExternalEnableFlag	bool[] **ExternalEnableFlag**() Returns true if the external enable input is high.
ExternalInhibitFlag	bool[] **ExternalInhibitFlag**() Returns true if the external enable input is disabling the motor.
IdentModeActiveFlag	bool[] **IdentModeActiveFlag**() Returns true if the ident mode is active.
JoystickConnectedFlag	bool[] **JoystickConnectedFlag**() Returns true if the joystick is connected.
LimitNegativeFlag	bool[] **LimitNegativeFlag**() Returns true if the negative limit is active.
LimitPositiveFlag	bool[] **LimitPositiveFlag**() Returns true if the positive limit is active.
MotorOverTemperatureFlag	bool[] **MotorOverTemperatureFlag**() Returns true if the motor temperature is greater than 190 °C.
MotorShortFlag	bool[] **MotorShortFlag**() Returns true if a motor phase to phase or phase to ground short has been detected.
MotorStandbyFlag	bool[] **MotorStandbyFlag**() Returns true if the motor is stationary.
TargetVelocityReachedFlag	bool[] **TargetVelocityReachedFlag**() Returns true if the motor is at the target step frequency.
TemperatureSensorOpenFlag	bool[] **TemperatureSensorOpenFlag**() Returns true if the selected temperature sensor is open circuit.

TemperatureSensorShortedFlag	bool[] **TemperatureSensorShortedFlag**() Returns true if the selected temperature sensor is shorted (not applicable to thermocouple)

Example scripts

Add device, rename and set device properties

```
// Add device with serial number 20054-027
smd.Add("20054-027");

// Set name of device with serial number 20054-027 to MyDevice
smd.Name("20054-027","MyDevice");

// Select device with name MyDevice
smd.Select("MyDevice");

// Set the acceleration and deceleration rate in Hz/s
smd.Acceleration(100);
smd.Deceleration(100);

// Set the acceleration current
smd.AccelerationCurrent(1.044);

// Set the hold current
smd.HoldCurrent(0);

// Set the run current
smd.RunCurrent(0.5);

// Set the start frequency
smd.StartFrequency(10);

// Set the step frequency
smd.StepFrequency(1000);

// Set the frequency at which the drive transitions to full step
smd.MicrostepTransition(500);

// Set the micostep resolution
smd.Resolution(64);
```

Move motor and wait

```
// Select device with name MyDevice
smd.Select("MyDevice");

// Set mode to remote
smd.Mode(2);

// Iterate for 10 times
for (i = 0; i < 10; i++) {
   // Move motor +500 steps
    smd.MoveRelative(500);

   // Wait 2 seconds
    smd.DelaySeconds(2);

   // Move motor -500 steps
```

```
    smd.MoveRelative(-500);

    // Wait 5 seconds
    smd.DelaySeconds(5);

}
```

Get value of actual position counter and log to command line

```
// Select device with name MyDevice
smd.Select("MyDevice");

// Store actual position in a variable
pact = smd.ActualPosition();

// Log result to command line
smd.Log(pact[0]);
```

Check if the motor is in standby

```
// Select device with name MyDevice
smd.Select("MyDevice");

// Store status flags in a variable
status = smd.StatusFlags();

// Log a specific bit status of the selected device to the command line
if(status[0] = status[0] & (0x1 << 6)){
    smd.Log("Motor is in standby");
} else {
    smd.Log("Motor is running");
}
```

Remote interfaces

The SMD4 offers 3 communication interfaces, USB, Ethernet and Serial. The serial interface is dual mode, and can be configured for RS232 or RS485. Multiple interfaces can be connected and in use at the same time. All interfaces use a text-based communication protocol described in section Communications protocol.

This section may only be relevant if you are using terminal software or writing your own application. AML Device control software interacts with the SMD4 in the same way and requires no specialist knowledge to use. A C# API is also provided, allowing you to easily integrate communication with the SMD4 into your own C# .NET application.

USB

A reversible USB-C connector is provided.

The SMD4 appears as a virtual COM port on your PC, and will be assigned a designation such as "COM1". Each unique SMD4 that is connected to the PC will be assigned a new designation. Upon reconnecting a previously known SMD4, it will typically assume the same designation as when last used, however, this behaviour cannot be guaranteed.

The virtual COM port does not require any particular configuration to function correctly. Typical settings such as baud rate and parity are not applicable.

Most programming languages include easy built-in methods to read and write data to and from a COM port, for example, the SerialPort class in the System.IO.Ports namespace in C#, or pySerial in Python. Alternatively, use a terminal program such as Tera Term to manually type commands and see responses. Remember to set line termination which is a carriage return followed by line feed (in hexadecimal, 0x0D, 0x0A)

Serial

A pair of RJ45 connectors are provided allowing multiple devices to be bussed together if required. This is discussed in section RS232/RS485. Do not confuse these connectors with the Ethernet connector.

The serial interface can be configured for RS232 or RS485.

If connecting to a PC, a serial port is required. Modern PCs seldom include RS232/485 ports, so typically a USB adapter or add-in card is required. This appears on the PC as one or more COM ports.

Unlike the virtual COM port provided via the USB interface, the COM port must be configured correctly:

- Baud to match that set on the SMD4
- 1 stop bit
- No parity
- No flow control

Thereafter, the same notes as for USB apply.

Configure the mode of operation (RS232 or RS485) before plugging a connector in. Do not make changes to the mode without first disconnecting both connectors. Undefined behaviour will result if the SMD4 is connected to an RS485 network when in RS232 mode or vice-versa.

Ethernet

A standard 10/100M ethernet port is provided. It features Auto MDI-X so straight or crossover cable may be used.

The factory default setup enables DHCP, so under normal circumstances, this will assign the network configuration to the SMD4. Alternatively, use another interface, such as USB, to configure the ethernet port.

As with COM ports, most programming languages provide easy means to open a socket and communicate with network devices. Alternatively, use a terminal program such as PuTTY to manually type commands and see responses.

The SMD4 uses port 11312 for its text-based communication protocol. Only one connection at a time is supported, additional connections will be refused.

SSDP

The SMD4 is discoverable on the network using SSDP (Simple Service Discovery Protocol). This makes it possible to plug the SMD4 into a network, then discover its IP address and so make a connection to it. Alternatively, use another interface, for example, USB to discover the network configuration.

Supported search targets (queries)

All – Not SMD4 specific, all devices on the network supporting this command will respond including SMD4

- "ssdp:all"

JPNP Root Device, as above

- "upnp:rootdevice"

)evice – Specify domain name, device type and version to match the SMD4 and only SMD4s will respond:

- "urn:schemas-arunmicro-com:device:StepperMotorDrive:1"
- "urn:" SSDPR_SCHEMA_DOMAIN_NAME ":device:" SSDPR_SCHEMA_DEVICE_TYPE ":" SSDPR_SCHEMA_VER"

UUID – Specify a UUID and only one device having that UUID should respond. Use **this command** to get the UUID of your SMD4.

- "uuid:" UUID

Where:

SSDPR_SCHEMA_DOMAIN_NAME	"schemas-arunmicro-com"
SSDPR_SCHEMA_DEVICE_TYPE	"StepperMotorDrive"
SSDPR_SCHEMA_VER	"1"

Response

The SMD4 always responds in this way:

```
HTTP/1.1 200 OK\r\n
CACHE-CONTROL:max-age=120\r\n
DATE:\r\n
EXT:\r\n
LOCATION:http://10.0.97.6:80/desc.xml\r\n
SERVER:OS/version product/version\r\n
ST:uuid:28254095-7194-11ee-a857-44b7d0c76aa3\r\n
USN:uuid:28254095-7194-11ee-a857-44b7d0c76aa3::urn:schemas-arunmicro-com:device:StepperMotorDrive:1\r\n
\r\n
```

Note:

- Response always starts with http ok
- CACHE_CONTROL, DATE and EXT are not used. CACHE-CONTROL has a default value but no meaning
- LOCATION is a URL to an xml file containing information as required by UPnP. The SMD4 does not implement a webserver to allow this file to be retrieved, however the root of the URL gives the IP address of the device which may be used to address future communications.
- SERVER:OS not implemented
- ST; this echos the search target sent in the query
- USN: gives the uuid and device type info

Communications protocol

A simple text-based protocol is used. Commands are sent to the SMD4, checked and executed, and a response returned. Data are buffered on receipt and commands are evaluated and executed on a first in first out basis. Although not a requirement, it is usually easiest to send a command and evaluate the response before sending the next command.

Commands are in the form (Note that angle brackets are shown for clarity only, they are not part of the protocol):

```
<address prefix><mnemonic>,<argument 1>,<argument 2>,<argument n>…<CR><LF>
```

And responses are in the form:

```
<address prefix>,<SFLAGS>,<EFLAGS>,<data 1>,<data 2>,<data n>…<CR><LF>
```

If the command executed successfully, or:

```
<address prefix>,<SFLAGS>,<EFLAGS>,<error code><CR><LF>
```

If the command failed to execute correctly.

Where:

Item	Description
<address prefix>	**Optional** prefix included when multiple SMD4s exist on the same bus. If not using addressing can be omitted.
<mnemonic>	Short sequence of characters representing a command, case insensitive
<argument n>	Zero or more command arguments
<data n>	Zero or more response data items
<error code>	An error code, see section Error Codes. This includes both a number and text description of the error to aid when using the SMD4 via a terminal program.
<SFLAGS>	Set of flags representing the status of the SMD4, such as the state of the limit inputs or whether the joystick is connected. See section Status Flags
<EFLAGS>	Set of flags representing the error state of the SMD4, such as invalid mnemonic, or motor over-temperature fault.
<CR><LF>	Message terminator; carriage return followed by line-feed (0x0D,0x0A)

Addressing

This section is only applicable where multiple SMD4s are connected together on the same bus, using the serial interface in either RS232 or RS485 mode. The addressing logic described in this section works for all interfaces, but is redundant for USB and the network interface since those inherently implement addressing.

When multiple SMD4s exist on the same bus, a mechanism is required to allow them to be addressed uniquely or as a group. Likewise, only one device must use the bus at a time otherwise bus contention results when more than one device tries to drive the bus at a time.

This is accomplished via the address prefix, which is the at '@' symbol followed by a numeric address:

- 0 = Broadcast address, all SMD4s execute the command, but no response is sent

- 1 to 247 = Valid secondary address range. The addressed SMD4 executes the command and returns a response
- Any address outside this range is invalid, and the packet is silently ignored

Upon receipt of the first complete packet with an address prefix, the SMD4 enters addressing mode, and behaviour then changes as follows, until restart.

- Malformed packets are silently ignored. This includes any packet that does not include the addressing prefix but that is otherwise valid.
- Broadcast packets are silently parsed and executed. A response is not sent, and as such it cannot be determined whether the command executed successfully without submitting a further query addressed directly to the target SMD4.
- Packets that are otherwise correctly formed but having a target address that does not match that of the SMD4 are silently ignored.

Comma separation

All elements are comma-separated, except for the message terminator which immediately follows the last item. A response is always sent on receipt of a message terminator except where addressing criteria are not met. If an argument was supplied with a command, for example, to set a value, the value set will be returned in the response and serves as an additional confirmation of the command having executed as expected.

Many commands accept a real number argument when the underlying quantity is an integer, or finite set of real numbers. In this case, the supplied value being otherwise acceptable is rounded to the closest integer or real number from the allowed set, and it is this value that is returned in the response.

No data items to return
If there are no data items as part of a response, only the SFLAGS and EFLAGS are returned. If an error occurred, then this will be reflected in the EFLAGS.

Argument types

Arguments may be one or a mix of the following types, depending on the command. Data returned by the SMD4 uses the same types, which are always presented as indicated in the “SMD4 response” column.

Type	Name	Description	Example argument values	SMD4 response
INT	Integer	Integer value, with or without sign	100, -10, +7	Sign included for negative values only. E.g. 100, -10
UINT	Unsigned integer	Unsigned integer value, no sign. Hexadecimal representation may also be used, case insensitive	99, 1000, 0xA74F, 0xd7	Numeric format. E.g. 100, 200 Except for status and error flags which are returned in upper case 2-byte hexadecimal format, E.g. 0x1234, 0xA4DE
FLOAT	Real number	Real number, with or without sign. Scientific format may also be used, case insensitive	10.23, 100e-3, 100E4, 10	Scientific format, with 5 places after the decimal point and a 2-digit exponent E.g. 1.23000E+04, 5.76159E-10
STRING	ASCII string	ASCII string, consisting of characters 0x20 to 0x7E inclusive	Abc123, 78-%^A	ASCII string, E.g. "1234 abc", "10%"

BOOL	Boolean	Binary, true/false value	0, 1	E.g. 0, 1
DOTTED DECIMAL	Dotted decimal	IPV4 address or mask, four numbers separated by dots.	192.168.0.1	E.g. an IP address 192.168.0.1 or a net mask 255.255.255.100

Flags

Error flags are reported by the device in hexadecimal format as explained above. E.g. a value of 0x0002 means bit 1 is set (TOPEN), indicating that the device has been disabled due to an open circuit temperature sensor.

Error flags (EFLAGS)

These indicate error conditions and are latching (i.e. remain set even after the error condition that caused them no longer persists). Reset the fault using the clear command, or the reset fault input. The motor is disabled if one or more error flags are set.

Bit	Name	Description
0	Temp Short	Selected temperature sensor is short-circuited (Not applicable to Thermocouple)
1	Temp Open	Selected temperature sensor is open circuit
2	Temp Over	Selected temperature sensor is reporting temperature > 190 °C and power has been removed from the motor to protect the windings
3	Motor Short	Motor phase to phase or phase to ground short has been detected
4	External Disable	Motor disabled via external input
5	Emergency Stop	Motor disabled via software
6	Configuration Error	Motor configuration is corrupted
7	Encoder error	Encoder fault (applicable only when optional encoder module installed)
8	Boost UVLO	The internal 48 V to 67 V boost circuit is disabled because input voltage has fallen too low.
9	SDRAM	Memory self-test failed.
10-15	Reserved	Reserved, read as '0'

Status flags (SFLAGS)

Bit	Name	Description
0	Joystick Connected	Joystick is connected (determined via state of the
1	Limit Negative	Limit input is active (Note that the polarity is configurable, so active can mean high or low signal level)

2	Limit Positive	Limit input is active (Note that the polarity is configurable, so active can mean high or low signal level)
3	External Enable	External enable input state
4	Ident	Ident mode is active, green status indicator is flashing to aid in identifying device
5-6	Reserved	Reserved, read as '0'
7	Standby	Motor stationary. Check this bit before performing a function that requires the motor to be stopped first, such as changing mode
8	Baking	Bake mode running
9	Target Velocity Reached	Set when the motor is at target velocity
10	Encoder Present	Encoder module fitted
11	Boost Operational	Internal 48 V to 67 V boost supply is operational
12-15	Reserved	Reserved, read as '0'

Error codes

Error	Description
-1 (Stop motor first)	Several actions, such as changing resolution or operating mode require that the motor is stopped first. Trying to run such a command before the motor has come to a stop and the standby flag in the status register is set will result in this error.
-2 (Argument validation)	An argument supplied to the command is invalid, for example, it is outside the allowable range.
-3 (Unable to get)	The command is write-only, read is not valid. This applies to commands such as RUNV where a read would have no meaning.
-5 (Action failed)	The command failed to execute due to an internal error, for example, the internal flash in which settings are stored has reached the end of life and data cannot be reliably written to it.
-6 (Not possible in mode)	The command is not applicable to this mode, for example, trying to start bake using RUNB when not in bake mode.
-7 (Not possible when motor disabled)	The motor is disabled (due to a fault, or external enable) and the command is one that starts motion, for example RUNV.
-101 (Argument type)	The argument is of the wrong type, for example a non-integer value was given where an integer value was required.
-102 (Argument count)	The argument count is incorrect, either too few or too many arguments have been supplied.

-103 (Invalid Mnemonic)	Command mnemonic is not valid
-104 (Packet error)	Packet is malformed

Quick reference

General

Mnemonic	Description	R	W	Arguments
SYS:IDENT	Rapidly blinks status indicator	●	●	BOOL
SYS:MODE	Mode of operation	●	●	UINT
SYS:JSMODE	Joystick mode	●	●	UINT
SYS:AUTOJS	Auto switch to Joystick mode on JS connect	●	●	BOOL
SYS:EXTEN	External enable used	●	●	BOOL
SYS:CLR	Clear error flags		●	
SYS:FLAGS	Get status and error flags	●		
SYS:FLAGSV	Get human readable summary of status and error flags	●		
SYS:FW	Read the firmware version number	●		STRING
SYS:LOAD	Load saved configuration		●	
SYS:LOADFD	Load factory default settings		●	
SYS:PROG	Enter programming mode		●	
SYS:RESET	Restart the SMD4		●	
SYS:BSN	Get motherboard serial number	●		STRING
SYS:PSN	Get product serial number	●		STRING
SYS:UPTIME	Get uptime	●		UINT
SYS:UUID	Get UUID	●		

Command movement

Mnemonic	Description	R	W	Arguments
MOTOR:RUNV	Move motor velocity mode		●	STRING
MOTOR:RUNA	Move motor absolute positioning mode		●	INT
MOTOR:RUNR	Move motor relative positioning mode		●	INT

MOTOR:RUNH	Start home mode procedure		●	STRING
MOTOR:STOP	Bring motor to a stop according to the current profile		●	
MOTOR:SSTOP	Stop motor in 1 second on full step position independently of the current motion profile		●	
MOTOR:ESTOP	Emergency stop. Stops the motor immediately		●	

Motor

Mnemonic	Description	R	W	Arguments
MOTOR:TSEL	Temperature sensor selection, T/C or RTD	●	●	UINT
MOTOR:T	Temperature in °C	●		
MOTOR:IR	Run current in amps	●	●	FLOAT
MOTOR:IA	Acceleration current in amps	●	●	FLOAT
MOTOR:IH	Hold current in amps	●	●	FLOAT
MOTOR:PDDEL	Power down delay in milliseconds	●	●	FLOAT
MOTOR:IHD	Delay per current reduction step	●	●	FLOAT
MOTOR:F	Freewheel mode	●	●	UINT
MOTOR:RES	Resolution	●	●	UINT
MOTOR:SDMODE	Step/direction mode	●	●	

Limit inputs

Mnemonic	Description	R	W	Arguments
LIMIT:EN	Global enable	●	●	BOOL
LIMIT:EN+	Limit positive (Limit 1) enable	●	●	BOOL
LIMIT:EN-	Limit negative (Limit 2) enable	●	●	BOOL
LIMIT:POL-	Limit n polarity (0 for active high, 1 for active low)	●	●	BOOL
LIMIT:POL+	Limit n polarity (0 for active high, 1 for active low)	●	●	BOOL

LIMIT:POL	Limit polarity for both Limit positive (Limit 1) and negative (Limit 2), (0 for active high, 1 for active low)		●	BOOL
LIMIT:STOPMODE	How to stop on limit being triggered	●	●	BOOL

Profile

Mnemonic	Description	R	W	Arguments
MOTOR:AMAX	Acceleration in Hz/s	●	●	FLOAT
MOTOR:DMAX	Deceleration in Hz/s	●	●	FLOAT
MOTOR:VSTART	Start frequency in Hz	●	●	FLOAT
MOTOR:VSTOP	Stop frequency in Hz	●	●	FLOAT
MOTOR:VMAX	Target step frequency in Hz	●	●	FLOAT
MOTOR:VACT	Actual frequency in Hz	●		
MOTOR:PACT	Actual position in steps	●	●	FLOAT
MOTOR:PREL	Relative position in steps	●	●	FLOAT
TZW	Time to stop before moving again in seconds	●	●	FLOAT
MOTOR:THIGH	Full step – micro stepping transition	●	●	FLOAT

Step/Direction

Mnemonic	Description	R	W	Arguments
MOTOR:EDGE	Which edges of step input to generate a step on	●	●	UINT
MOTOR:INTERP	Interpolate step input to 256 micro steps	●	●	BOOL

Bake

Mnemonic	Description	R	W	Arguments
BAKE:T	Bake temperature setpoint	●	●	UINT
BAKE:RUN	Start bake		●	
BAKE:ELAPSED	Get the elapsed bake time		●	

Boost

Mnemonic	Description	R	W	Arguments
BOOST:EN	Boost enable	●	●	BOOL

Coms

Mnemonic	Description	R	W	Arguments
COMS:NET:DHCP	Gets or sets a value indicating whether DHCP is enabled	●	●	BOOL
COMS:NET:GATEWAY	Gets or sets the gateway address	●	●	DOTTED DECIMAL
COMS:NET:NETMASK	Gets or sets the subnet mask	●	●	DOTTED DECIMAL
COMS:NET:IP	Gets or sets the IP address	●	●	DOTTED DECIMAL
COMS:NET:IPCONF	Outputs a summary of network configuration in human readable form	●		
COMS:NET:LINK	Gets a value indicating whether the ethernet interface link is up	●		BOOL
COMS:NET:MAC	Gets the Ethernet interface MAC address	●		
COMS:SERIAL:BAUD	Gets or sets the baud rate	●	●	UINT
COMS:SERIAL:MODE	Gets or sets the serial coms mode, either RS232 or RS485	●	●	
COMS:SERIAL:RS485DI	Gets or sets a value in milliseconds specifying the delay to execute between receipt of a command from the host and the client (SMD4) sending the response	●	●	UINT
COMS:SERIAL:TERM	Gets or sets a value indicating whether RS485 line termination should be used	●	●	BOOL
COMS:SERIAL:SLAVEAI	Gets or sets the slave address	●	●	UINT

Command reference

SYS:IDENT - Rapidly blinks status indicator (R/W)

Gets or sets a value indicating whether the identify function is enabled. When set to true, the green status light on the front of the product flashes. This can be used to help identify one device amongst several.

Command: **SYS:IDENT**, **Enable**<CR><LF>
Query: **SYS:IDENT**<CR><LF>

Arguments

Enable	BOOL

The enable state.

[0:	Disable]
1:	Enable

Returns

The enable state, as above.

Examples

```
Tx: SYS:IDENT,1<CR><LF>                    // Set ident function on
Rx: 0x0000,0x0000,1<CR><LF>
Tx: SYS:IDENT<CR><LF>                      // Query state of ident function
Rx: 0x0000,0x0000,1<CR><LF>
```

SYS:MODE - Choose mode of operation

Gets or sets the operating mode. See sectionOperating Modes for an explanation of each mode.

Command: **SYS:MODE**, **Value**<CR><LF>
Query: **SYS:MODE**<CR><LF>

Arguments

Value	UINT

The operating mode.

0:	Step/direction
[1:	Remote]
2:	Joystick
3:	Bake
4:	Home

Returns

The mode, as above, followed by a space and the name of the mode in brackets.

Remarks

If the motor is moving when attempting to change the mode, a stop motor first error is returned and the mode is unchanged.

Examples

```
Tx: SYS:MODE,2<CR><LF>                     // Set mode to remote
Rx: 0x0000,0x0000,2 (Remote)<CR><LF>
Tx: SYS:MODE<CR><LF>                       // Query state of mode
Rx: 0x0000,0x0000,1 (Remote)<CR><LF>
```

SYS:JSMODE – Joystick mode

Gets or sets the joystick mode. Choose between single step, which allows precise single steps or continuous rotation, or continuous which requires only a single button press to make the motor move.

Command: **SYS:JSMODE, Mode**<CR><LF>
Query: **SYS:JSMODE**<CR><LF>

Arguments

Mode	UINT

The joystick mode.

[0:	Single step]
1:	Continuous

Returns

The mode, as above.

Remarks

Set requires the motor to be in standby, otherwise, a stop motor first error will be returned.

In single step mode, a brief button press (< 0.5 s) will execute one step in that direction, while pressing the button for > 0.5 s will cause the motor to accelerate up to slewing speed and continue to rotate in that direction until the button is released, at which point the motor will decelerate to a stop.

In continuous mode, a brief button press will trigger the motor to accelerate up to slewing speed. A subsequent press of the same button causes it to decelerate to a stop. If, for example, the clockwise button is pressed while the motor is rotating anti-clockwise, the motor will first decelerate to a stop before changing direction.

Examples

```
Tx: SYS:JSMODE,1<CR><LF>                    // Set to continuous
Rx: 0x0000,0x0000,1<CR><LF>
Tx: SYS:JSMODE<CR><LF>                      // Query state
Rx: 0x0000,0x0000,1<CR><LF>
```

SYS:AUTOJS – Auto switch to joystick mode

Gets or sets the joystick auto select function. When set to true, the product switches to joystick mode automatically when connecting a joystick.

Command: **SYS:AUTOJS, Enable**<CR><LF>
Query: **SYS:AUTOJS**<CR><LF>

Arguments

Enable	BOOL

The enable state.

0:	Disable
[1:	Enable]

Returns

The enable state, as above.

Examples

```
Tx: SYS:AUTOJS,1<CR><LF>                                // Enable
Rx: 0x0000,0x0000,1<CR><LF>
Tx: SYS:AUTOJS<CR><LF>                                  // Query state
Rx: 0x0000,0x0000,1<CR><LF>
```

SYS:EXTEN – External enable used

Gets or sets a value indicating whether the external enable signal should be respected. If not using the external enable and it remains disconnected, set to false.

Command: **SYS:EXTEN, Used**<CR><LF>
Query: **SYS:EXTEN**<CR><LF>

Arguments

Used	BOOL

External enable signal.

[0:	False]
1:	True

Returns

True if the external enable signal is used.

Remarks

The external enable input requires a voltage to be applied between SDE COM and EN on the I/O connector which may be inconvenient if you do not wish to use the enable input. In that case, disable the enable input by sending this command with the argument set to false.

Examples

```
Tx: SYS:EXTEN,1<CR><LF>                                 // Enable
Rx: 0x0000,0x0000,1<CR><LF>
Tx: SYS:EXTEN<CR><LF>                                   // Query state
Rx: 0x0000,0x0000,1<CR><LF>
```

SYS:CLR – Clear faults

Clear all error flags.

Command: **SYS:CLR**<CR><LF>

Examples

```
Tx: SYS:CLR<CR><LF>                                     // Clear errors
Rx: 0x0000,0x0000<CR><LF>
```

SYS:FLAGS – Get status and error flags

Gets status and error flags.

Query: **SYS:FLAGS**<CR><LF>

Remarks

None.

Examples

```
Tx: SYS:FLAGS<CR><LF>                                    // Get flags
Rx: 0x0000,0x0000,1<CR><LF>
```

SYS:FLAGSV – Get status and error flags summary

Gets a human readable summary of status and error flags.

Query: **SYS:FLAGSV**<CR><LF>

Remarks

None.

Examples

```
Tx: SYS:FLAGSV<CR><LF>
Rx: 0x088e,0x0000,<CR><LF>

-------Status flags------
[ ]JsCon
[X]LimitNeg
[X]LimitPos
[X]Exten
[ ]Ident
[ ]reserved1
[ ]reserved2
[X]Standby
[ ]Baking
[ ]TargetVelocityReached
[ ]EncoderPresent
[X]BoostOperational
[ ]BoostDisableJumper
[ ]reserved3
[ ]reserved4
[ ]reserved5

-------Error flags-------
[ ]TempShort
[ ]TempOpen
[ ]TempOver
[ ]MotorShort
[ ]ExternalInhibit
[ ]EmergencyStop
[ ]ConfigError
[ ]EncoderError
[ ]BoostUVLO
[ ]reserved1
[ ]reserved2
[ ]reserved3
[ ]reserved4
[ ]reserved5
[ ]reserved6
[ ]reserved7
```

SYS:FW – Get firmware version

Gets firmware version string.

Query: **SYS:FW**<CR><LF>

Returns

Firmware version STRING

Remarks

None.

Examples

```
Tx: SYS:FW<CR><LF>                                    // Query
Rx: 0x088e,0x0000,24044.12<CR><LF>
```

SYS:LOAD – Load last stored settings

Load the last saves configuration.

Command: **SYS:LOAD**<CR><LF>

Remarks

None.

Examples

```
Tx: SYS:LOAD<CR><LF>
Rx: 0x088e,0x0000<CR><LF>
```

SYS:LOADFD – Load factory default settings

Load the factory default configuration.

Command: **SYS:LOADFD**<CR><LF>

Remarks

Use the store command if you want to persist the changes.

Examples

```
Tx: SYS:LOADFD<CR><LF>
Rx: 0x088e,0x0000<CR><LF>
```

SYS:PROG – Enter programming mode

Reboot the SMD4 into programming mode. Used by AML device control software to initiate a firmware update. Power cycle to cancel this mode.

Command: **SYS:PROG**<CR><LF>

Remarks

There is no response to this command.

Examples

```
Tx: SYS:PROG<CR><LF>
```

SYS:RESET– Restart the SMD4

Reboot the SMD4.

Command: **SYS:RESET**<CR><LF>

Remarks

There is no response to this command.

Examples

```
Tx: SYS:RESET<CR><LF>
```

SYS:BSN – Get motherboard serial number

Gets the serial number of the motherboard.

Query: **SYS:BSN**<CR><LF>

Returns

Motherboard serial number STRING

Remarks

None.

Examples

```
Tx: SYS:BSN<CR><LF>                                  // Query
Rx: 0x088e,0x0000,1234ABCD<CR><LF>
```

SYS:PSN – Get product serial number

Gets the serial number of the product. This matches the serial number label installed on the product.

Query: **SYS:PSN**<CR><LF>

Returns

Product serial number STRING

Remarks

None.

Examples

```
Tx: SYS:PSN<CR><LF>                                  // Query
Rx: 0x088e,0x0000,00000-000<CR><LF>
```

SYS:UPTIME – Get uptime

Gets the elapsed time since start up in milliseconds.

Query: **SYS:UPTIME**<CR><LF>

Returns

Uptime UINT

Remarks

None.

Examples

```
Tx: SYS:UPTIME<CR><LF>                        // Query
Rx: 0x088e,0x0000,10000<CR><LF>               // Uptime is 10 seconds
```

SYS:UUID – Get UUID

Gets a unique ID number which is included in the data reported when using SSDP. See SSDP. Not the same as the MAC address.

Query: **SYS:UUID**<CR><LF>

Returns

UUID UUID

Remarks

None.

Examples

```
Tx: SYS:UUID<CR><LF>                          // Query
Rx: 0x088e,0x0000,f4562fb1-d002-11ee-b3e5-
44b7d0c71675<CR><LF>
```

Command movement

MOTOR:RUNV – Run, velocity

Start continuous rotation in specified direction.

Command: **MOTOR:RUNV, Direction** <CR><LF>

Arguments

Direction	String

Direction:

'+':	Positive, step count increases
'-':	Negative, step count decreases

Remarks

None.

Examples

```
Tx: MOTOR:RUNV,+<CR><LF>                      // Spin motor in positive direction
Rx: 0x0000,0x0000<CR><LF>
Tx: MOTOR:RUNV,-<CR><LF>                      // Spin motor in negative direction
Rx: 0x0000,0x0000<CR><LF>
```

MOTOR:RUNA – Run, absolute position

Move the motor to a specified absolute position.

Command: **MOTOR:RUNA, Absolute** <CR><LF>

Arguments

Absolute	INT
Minimum:	-8388608

Maximum:	8388607

Remarks

None.

Examples

```
Tx: MOTOR:RUNA,1000<CR><LF>                 // Drive motor to step position 1000
Rx: 0x0000,0x0000<CR><LF>
Tx: MOTOR:RUNA,-1000<CR><LF>                // Drive motor to step position -1000
Rx: 0x0000,0x0000<CR><LF>
```

MOTOR:RUNR - Run, relative position

Move the motor a specified number of steps, relative to the current position.

Command: **MOTOR:RUNR, Relative** <CR><LF>

Arguments

Relative	INT
Minimum:	-8388608
Maximum:	8388607

Remarks

None.

Examples

```
Tx: MOTOR:RUNR,2000<CR><LF>                 // Move motor in positive direction by 2000 steps
Rx: 0x0000,0x0000,1<CR><LF>
Tx: MOTOR:RUNR,-2000<CR><LF>                // Move motor in negative direction by 2000 steps
Rx: 0x0000,0x0000,1<CR><LF>
```

MOTOR:RUNH - Run, home

Initiate a homing sequence to the specified limit.

Command: **MOTOR:RUNH, Direction**<CR><LF>

Arguments

Direction	String

Direction:

‘+’:	Home towards positive limit, step count increases
‘-’:	Home towards negative, step count decreases

Remarks

None.

Examples

```
Tx: MOTOR:RUNR,2000<CR><LF>                 // Move motor in positive direction by 2000 steps
Rx: 0x0000,0x0000,1<CR><LF>
Tx: MOTOR:RUNR,-2000<CR><LF>                // Move motor in negative direction by 2000 steps
Rx: 0x0000,0x0000,1<CR><LF>
```

MOTOR:STOP – Stop motor

Stop the motor, decelerating according to the current profile

Command: **MOTOR:STOP**<CR><LF>

Remarks

During the deceleration phase that stops the motor, any modifications to the acceleration or deceleration interrupt the stopping phase. Re-send the command to restart the motor stopping phase.

Examples

```
Tx: MOTOR:STOP<CR><LF>                                // Stop the motor
Rx: 0x0000,0x0000<CR><LF>
```

MOTOR:SSTOP – Stop motor in <=1 s

Decelerates the motor to a stop within 1 second, disregarding the current profile to do so.

Command: **MOTOR:SSTOP**<CR><LF>

Remarks

This command does not consider the deceleration set in the profile. Instead, it calculates the deceleration required to stop in 1 second, according to the actual velocity. The motor will stop in a full step position. Steps may be lost if the load requires greater than this duration to stop.

Examples

```
Tx: MOTOR:SSTOP<CR><LF>                               // Stop the motor in 1 seconds
Rx: 0x0000,0x0000<CR><LF>
```

MOTOR:ESTOP – Emergency stop

Stop motor immediately disregarding deceleration profile and disable the motor. This should not be relied on as a safety interlock.

Command: **MOTOR:ESTOP**<CR><LF>

Remarks

The motor may stop on a fractional step position, but this is irrelevant as motor power is removed and the motor will snap to a full step position. Steps may be lost.

Examples

```
Tx: MOTOR:ESTOP<CR><LF>                               // Stop the motor immediately
Rx: 0x0000,0x0000<CR><LF>
```

Motor

MOTOR:TSEL – Temperature sensor selection

Gets or sets the motor temperature sensor type.

Command: **MOTOR:TSEL, SensorType**<CR><LF>

Query: **TSEL**<CR><LF>

Arguments

SensorType	UINT

Motor temperature sensor type.

[0:	Thermocouple]
1:	RTD

Returns

Selected temperature sensor type, as above.

Remarks

To protect the motor from possible damage, the motor is disabled if the temperature sensor is faulty or missing. The response is not immediate, and several seconds may elapse between emergence of a fault and the motor being disabled.

Examples

```
Tx: MOTOR:TSEL,0<CR><LF>              // Select thermocouple sensor
Rx: 0x0000,0x0000,0<CR><LF>
Tx: MOTOR:TSEL<CR><LF>                // Get selected sensor, returning 0 for thermocouple
Rx: 0x0000,0x0000,0<CR><LF>
```

MOTOR:T – Motor temperature

Get the motor temperature in °C.

Query: **MOTOR:T**<CR><LF>

Returns

Motor temperature as integer in °C.

Remarks

The reported temperature is intended only for the purposes of monitoring motor temperature and should not be relied upon for any other purpose within the vacuum system.

Examples

```
Tx: MOTOR:T<CR><LF>                   // Get motor temperature
Rx: 0x0000,0x0000,25<CR><LF>          // Response is 25 degrees Celsius
```

MOTOR:IR – Run current

Gets or sets the motor run current.

Command: **MOTOR:IR, Current**<CR><LF>
Query: **MOTOR:IR**<CR><LF>

Arguments

Current	FLOAT

The motor run current in amps rms.

[Default:	1.044]
Minimum:	0
Maximum:	1.044

Returns

The motor run current in amps rms, rounded to the closest multiple of 1.044 A / 31 (approx. 33 mA).

Remarks

Run current must be set equal to or smaller than acceleration current. Acceleration current is automatically adjusted to be equal to run current, if a change to run current makes it greater than acceleration current.

Examples

```
Tx: MOTOR:IR,1<CR><LF>                        // Set run current to 1 A
Rx: 0x0000,0x0000,1.0000E+00<CR><LF>
Tx: MOTOR:IR<CR><LF>                          // Query run current
Rx: 0x0000,0x0000,1.0000E+00<CR><LF>
```

MOTOR:IA – Acceleration current

Gets or sets the motor current applied during acceleration or deceleration.

Command:	**MOTOR:IA, Current**<CR><LF>
Query:	**MOTOR:IA**<CR><LF>

Arguments

Current	FLOAT

The motor acceleration current in amps rms.

[Default:	1.044]
Minimum:	0
Maximum:	1.044

Returns

The motor acceleration current in amps rms, rounded to the closest multiple of 1.044 A / 31 (approx. 33 mA).

Remarks

Acceleration current must be set equal to or greater than run current. Acceleration current is not adjusted to match run current if acceleration current is smaller than run current.

Examples

```
Tx: MOTOR:IA,1.044<CR><LF>                    // Set acceleration current to 1.044 A
Rx: 0x0000,0x0000,1.0440E+00<CR><LF>
Tx: MOTOR:IA<CR><LF>                          // Query acceleration current
Rx: 0x0000,0x0000,1.0440E+00<CR><LF>
```

MOTOR:IH – Hold current

Set or query the motor hold current. If your application allows it, setMOTOR:PDDEL, MOTOR:IHD and MOTOR:IH to zero in order to reduce run current to zero as quickly as possible after stopping which minimises motor temperature rise.

Command:	**MOTOR:IH, Current**<CR><LF>
Query:	**MOTOR:IH**<CR><LF>

Arguments

Current	FLOAT

The motor hold current in amps rms.

[Default:	0.1]
Minimum:	0
Maximum:	1.044

Returns

The motor hold current in amps rms, rounded to the closest multiple of 1.044 A / 31 (approx. 33 mA).

Examples

```
Tx: MOTOR:IH,0.5<CR><LF>                          // Set hold current to 0.5 A
Rx: 0x0000,0x0000,5.0000E-01<CR><LF>
Tx: MOTOR:IH<CR><LF>                              // Query hold current
Rx: 0x0000,0x0000,5.0000E-01<CR><LF>
```

MOTOR:PDDEL – Power down delay

Gets or sets the delay time in seconds between stand still occurring and the motor current being reduced from the acceleration current to the hold current. The range is 0 to 5.5 seconds, with approximately 8 bit / 20 ms resolution. See also DelayPerCurrentReductionStep.

Refer to Figure 1. If your application allows it, setMOTOR:PDDEL, MOTOR:IHD and MOTOR:IH to zero in order to reduce run current to zero as quickly as possible after stopping which minimises motor temperature rise.

Command: **MOTOR:PDDEL, Duration**<CR><LF>
Query: **MOTOR:PDDEL**<CR><LF>

Arguments

Duration	FLOAT

The power-down delay in seconds.

[Default:	0]
Minimum:	0
Maximum:	5.5

Returns

The power-down delay rounded to the closest settable value.

Examples

```
Tx: MOTOR:PDDEL,100E-3<CR><LF>                    // Set to 100 ms
Rx: 0x0000,0x0000,1.0000E-01<CR><LF>
Tx: MOTOR:PDDEL<CR><LF>                           // Query
Rx: 0x0000,0x0000,1.0000E-01<CR><LF>
```

MOTOR:IHD – Delay per current reduction step

Gets or sets the delay in seconds per current reduction step that occurs when run current is reduced to hold current. Non-zero values result in a smooth reduction in current which reduces the chance of a jerk upon power down. The range is 0 to 328 ms, with a resolution of 4 bits or approx. 20 ms. Current setting has a resolution of 5 bits, or 32 steps, and consequently the current reduction process will only have as many steps as exist between the configured run and hold current. See also MOTOR:PDDEL.

See Figure 1. If your application allows it, setMOTOR:PDDEL, MOTOR:IHD and MOTOR:IH to zero in order to reduce run current to zero as quickly as possible after stopping which minimises motor temperature rise.

Command: **MOTOR:IHD, Duration**<CR><LF>

Query: **MOTOR:IHD**<CR><LF>

Arguments

Duration	FLOAT

The delay per current reduction step in seconds.

[Default:	0]
Minimum:	0
Maximum:	328 ms

Returns

The delay per current reduction step in seconds.

Remarks

See also section Going to standby

Examples

```
Tx: MOTOR:IHD,328E-3<CR><LF>                        // Set IHD to 328 ms
Rx: 0x0000,0x0000,3.2800E-01<CR><LF>
Tx: MOTOR:IHD<CR><LF>                               // Query IHD
Rx: 0x0000,0x0000,3.2800E-01<CR><LF>
```

MOTOR:F – Freewheel mode

Gets or sets the freewheel mode. For maximum passive braking use phases shorted. Use freewheel to electrically disconnect the motor and allow it to freewheel. Hold current must be set to zero for this option to work.

The chosen mode becomes active after a time period specified by ‘PDDEL’ and ‘IHD’

Command: **MOTOR:F, Mode**<CR><LF>

Query: **MOTOR:F**<CR><LF>

Arguments

Mode	UINT

The freewheel mode:

0:	Normal
1:	Freewheel
[2:	Phases shorted to GND]

Returns

The freewheel mode selection, as above.

Remarks

Use the freewheel mode to allow the motor shaft to spin freely when the motor current is zero. The phases shorted to GND option supplies no power to the motor, but by shorting the phases together a holding torque is produced, and the motor shaft offers considerable resistance to movement. This is enough in many applications to remove the need for any holding current, with the benefit that no heat is generated because the motor phases are not energised.

Examples

```
Tx: MOTOR:F,1<CR><LF>                               // Set to freewheel mode
Rx: 0x0000,0x0000,1<CR><LF>                         // motor shaft can be turned easily
Tx: MOTOR:F<CR><LF>
Rx: 0x0000,0x0000,1<CR><LF>                         // Query
```

MOTOR:RES - Resolution

Gets or sets the microstep resolution.

Command: **MOTOR:RES, Resolution**<CR><LF>

Query: **MOTOR:RES**<CR><LF>

Arguments

Resolution	UINT

The microstep resolution as an integer.

[Default:	256]
Possible values:	8, 16, 32, 64, 128, 256

Returns

The microstep resolution, as above.

Remarks

Motor must be in standby to set the resolution.

The resolution applies globally, including for the step/direction interface. Each step on the step/direction interface generates a 1/8, 1/16, 1/32 etc step according to the resolution set here.

Above a configurable step frequency, the drive switches from the microstepping resolution specified here to full step mode in any case. See section THIGH

Examples

```
Tx: MOTOR:RES,256<CR><LF>                    // Set resolution to 256
Rx: 0x0000,0x0000,256<CR><LF>
Tx: MOTOR:RES<CR><LF>                        // Query
Rx: 0x0000,0x0000,256<CR><LF>
```

MOTOR:SDMODE - Step/direction mode

Gets of sets the step/direction mode. In normal mode, edges on the step input generate steps according to the edge setting, see <edge>. In triggered mode, continuous motion is triggered by an edge on the step input; this is akin to how continuous mode works for the joystick, see <joystick continuous mode>

Command: **MOTOR:SDMODE, Mode**<CR><LF>

Query: **MOTOR:SDMODE**<CR><LF>

Arguments

Mode	ENUM

Mode

[0:	Normal]
1:	Triggered

Returns

Mode ENUM

Remarks

None.

Examples

```
Tx: MOTOR:SDMODE,0<CR><LF>                   // Set mode 0, normal
Rx: 0x0000,0x0000,0<CR><LF>
```

Limit inputs

LIMIT:EN – Limits global enable

Gets or sets global limit enable state. If this setting is false, limits are disabled regardless of the state of any other limits configuration item

This does not affect other limits configuration settings, allowing limits to be configured as desired, then globally enabled or disabled if required.

Command:	**LIMIT:EN, Enabled**<CR><LF>
Query:	**LIMIT:EN**<CR><LF>

Arguments

Enabled	BOOL

Enable state of limits.

[0:	Disable]
1:	Enable

Returns

True if limits are globally enabled.

Remarks

This option globally enables or disabled limits; remaining limits settings remain unchanged.

Examples

```
Tx: LIMIT:EN,0<CR><LF>                          // Disable limits globally
Rx: 0x0000,0x0000,0<CR><LF>
Tx: LIMIT:EN<CR><LF>                            // Query
Rx: 0x0000,0x0000,0<CR><LF>
```

LIMIT:EN-, LIMIT:EN+ Negative limit enable, positive limit enable

Gets or sets the negative limit (corresponding to decrementing step counter), or positive limit (corresponding to incrementing step counter) enable.

Command:	**LIMIT:ENx,Enabled**<CR><LF>
Query:	**LIMIT:ENx**<CR><LF>

Where 'x' is '-' for negative or '+' for positive limit.

Arguments

Enabled	BOOL

Enable state of limit n.

0:	Disable
[1:	Enable]

Returns

True if limit is enabled.

Remarks

None.

Examples

```
Tx: LIMIT:EN+,1<CR><LF>                     // Set positive limit enable
Rx: 0x0000,0x0000,1<CR><LF>
Tx: LIMIT:EN-<CR><LF>                       // Query negative limit enable state
Rx: 0x0000,0x0000,1<CR><LF>
```

LIMIT:POL-, LIMIT:POL+ Negative limit polarity, positive limit polarity

Gets or sets the negative or positive limit polarity.

Command:	**LIMIT:POLx,Polarity**<CR><LF>
Query:	**LIMIT:POLx**<CR><LF>

Where 'x' is '-' for negative or '+' for positive limit.

Arguments

Polarity	UINT

Polarity of limit.

[0:	Active high]
1:	Active low

Returns

Polarity setting for the limit.

Remarks

None.

Examples

```
Tx: LIMIT:POL-,1<CR><LF>                    // Set negative limit polarity to active low
Rx: 0x0000,0x0000,1<CR><LF>
Tx: LIMIT:POL+<CR><LF>                      // Query positive limit polarity
Rx: 0x0000,0x0000,1<CR><LF>
```

LIMIT:POL – Global limit polarity

Set the polarity for both limits at once.

Command:	**LIMIT:POL,Polarity**<CR><LF>

Arguments

Polarity	UINT

Polarity of LP.

[0:	Active high]
1:	Active low

Remarks

None.

Examples

```
Tx: LIMIT:POL,1<CR><LF>                     // Set polarity of both limits to active low
Rx: 0x0000,0x0000,1<CR><LF>
```

LIMIT:STOPMODE – Limit stop mode

Gets or sets the limits stop mode, which determines behaviour on limit being triggered.

Command:	**LIMIT:STOPMODE, Mode**<CR><LF>
Query:	**LSM**<CR><LF>

Arguments

Mode	UINT

The stop mode.

[0:	Hard stop; the motor will stop immediately on a limit b triggered]
1:	Soft stop; the motor decelerates according to the profi

Returns

The stop mode, as above.

Remarks

When using hard stop, keep in mind that steps may be lost depending on the slewing speed and load on the motor. Treat position counters with caution until the true position has been established. Conversely, when using soft stop, ensure that the motor can decelerate to a stop before the physical end of travel is reached and steps are lost.

Examples

```
Tx: LIMIT:STOPMODE,1<CR><LF>          // Set soft stop mode
Rx: 0x0000,0x0000,1<CR><LF>
Tx: LIMIT:STOPMODE<CR><LF>            // Query
Rx: 0x0000,0x0000,1<CR><LF>
```

Profile

MOTOR:AMAX - Acceleration

Gets or sets the acceleration, in Hz/s (steps per second per second).

Command:	**MOTOR:AMAX, Acceleration**<CR><LF>
Query:	**MOTOR:AMAX**<CR><LF>

Arguments

Acceleration	FLOAT

The acceleration in Hz/s.

[Default:	5000]
Minimum:	10
Maximum:	15000

Returns

User value (data 1) and real value (data 2). See user/real values.

Remarks

None.

Examples

```
Tx: MOTOR:AMAX,150<CR><LF>                      // Set acceleration to 150Hz/s
Rx: 0x0000,0x0000,1.5000E+02,1.4988E+02<CR><LF> // Note that the target value of 150 has been adjusted to
Tx: AMAX<CR><LF>                                the closest real value, which deviates from the requested
Rx: 0x0000,0x0000,1.5000E+02,1.4988E+02<CR><LF> value by 0.12 Hz/s
```

MOTOR:DMAX - Deceleration

Gets or sets the deceleration, in Hz/s (steps per second per second).

Command:	**MOTOR:DMAX, Deceleration**<CR><LF>
Query:	**MOTOR:DMAX**<CR><LF>

Arguments

Deceleration	FLOAT

The deceleration in Hz/s.

[Default:	5000]
Minimum:	10
Maximum:	15000

Returns

User value (data 1) and real value (data 2). See user/real values.

Remarks

None.

Examples

```
Tx: MOTOR:DMAX,150<CR><LF>                      // Set deceleration to 150Hz/s
Rx: 0x0000,0x0000,1.5000E+02,1.4988E+02<CR><LF>
Tx: MOTOR:DMAX<CR><LF>                          // Query deceleration
Rx: 0x0000,0x0000,1.5000E+02,1.4988E+02<CR><LF>
```

MOTOR:VSTART – Start frequency

Get or set the start frequency in Hz. Must be set less than or equal toMOTOR:VSTOP. The acceleration ramp starts from this frequency.

The start frequency is the initial step rate, and helps to allow the motor to overcome inertia and start moving smoothly; if start frequency were zero, the duration of the initial few steps might be long enough that the motor would overcome inertia on the first step, then effectively stop for a time, then have to overcome inertia once more for the second step, and so on, until the steps were frequent enough that the motor remains moving.

Command:	**MOTOR:VSTART, StartFrequency**<CR><LF>
Query:	**MOTOR:VSTART**<CR><LF>

Arguments

StartFrequency	FLOAT

The start frequency in Hz.

[Default:	100]
Minimum:	1
Maximum:	700

Returns

User value (data 1) and real value (data 2). See user/real values.

Remarks

Start frequency must be set equal to or less than stop frequency. If a change to start frequency makes it bigger than stop frequency, stop frequency is automatically adjusted to be equal to start frequency.

Start frequency must be set equal to or less than step frequency. Start frequency is not adjusted to match step frequency if start frequency is greater than step frequency.

Examples

```
Tx: MOTOR:VSTART,0<CR><LF>                          // Set start frequency to 0 Hz
Rx: 0x0000,0x0000,0.0000+00,0.0000+00<CR><LF>
Tx: MOTOR:VSTART<CR><LF>                            // Query
Rx: 0x0000,0x0000,0.0000+00,0.0000+00<CR><LF>
```

MOTOR:VSTOP – Stop frequency

Get or set the stop frequency in Hz. Must be greater than or equal to MOTOR:VSTART. The deceleration ramp ends at this frequency. The final step before stop will occur at this frequency.

The stop frequency is the frequency at which the deceleration ramp ends; i.e. the deceleration ramp does not go from the target frequency linearly down to 0, but from the target frequency linearly down to the stop frequency.

Command:	**MOTOR:VSTOP, StopFrequency**<CR><LF>
Query:	**MOTOR:VSTOP**<CR><LF>

Arguments

StopFrequency	FLOAT

The stop frequency in Hz.

[Default:	100]
Minimum:	1
Maximum:	700

Returns

User value (data 1) and real value (data 2). See user/real values.

Remarks

Stop frequency must be set equal to or greater than start frequency. If a change to stop frequency makes it smaller than start frequency, start frequency is automatically adjusted to be equal to stop frequency.

Stop frequency must be set equal to or less than step frequency. Stop frequency is not adjusted to match step frequency if stop frequency is greater than step frequency.

Examples

```
Tx: MOTOR:VSTOP,10<CR><LF>                          // Set stop frequency to 10 Hz
Rx: 0x0000,0x0000,1.0000+01,9.9996+00<CR><LF>       // Notice the closest real value of 9.9996 Hz set
Tx: MOTOR:VSTOP<CR><LF>                             // Query
Rx: 0x0000,0x0000,1.0000+01,9.9996+00<CR><LF>
```

MOTOR:VMAX – Step frequency

Gets or sets the target step frequency in Hz, or steps per second. This is the maximum speed the motor will be run at. The target frequency will only be reached if there is enough time or distance to do so; if moving for a short time, for example, the motor may only accelerate to some fraction of the target frequency before it is time to decelerate to a stop.

Command:	**MOTOR:VMAX, TargetFrequency**<CR><LF>
Query:	**MOTOR:VMAX**<CR><LF>

Arguments

TargetFrequency	FLOAT

The target frequency in Hz.

[Default:	1 kHz]
Minimum:	1 Hz
Maximum:	15 kHz

Returns

User value (data 1) and real value (data 2). See user/real values.

Remarks

Motor torque decreases with speed, and each motor will have a different maximum frequency that it can achieve while reliably maintaining synchronicity (when synchronicity is lost, the motor fails to complete the steps that it is commanded to, leading to a difference between the true and actual positions), depending on the load it is driving.

Step frequency must be set equal to or greater than start frequency and stop frequency. Step frequency is not adjusted to match start frequency and stop frequency if step frequency is smaller than start frequency and stop frequency.

Examples

```
Tx: MOTOR:VMAX,1000<CR><LF>                          // Set step frequency to 1 kHz
Rx: 0x0000,0x0000,1.0000E+03,1.0000E+03<CR><LF>
Tx: MOTOR:VMAX<CR><LF>                               // Query
Rx: 0x0000,0x0000,1.0000E+03,1.0000E+03<CR><LF>
```

MOTOR:VACT – Actual frequency

Get the live step frequency of the motor in Hz (steps per second).

Query:	**MOTOR:VACT**<CR><LF>

Returns

The frequency at which the motor is spinning in Hz.

Remarks

This value is derived from the stepper motor control logic; there is no feedback from the motor itself. Hence, the motor could be stalled while which continues to indicate the expected.

Examples

```
Tx: MOTOR:VACT<CR><LF>                               // Query state of blink
Rx: 0x0000,0x0000,1.0000E+03<CR><LF>
```

MOTOR:PACT – Actual position

Gets or sets the actual position in steps.

The usual way to position the motor is to initialise the actual position to some reference value, usually 0, then adjust the target position to move the motor. In this way, by setting MOTOR:RUNA to 0 the motor can be homed to the initial 0 position. If you wish to perform relative movements, while still retaining an absolute reference, see MOTOR:PREL command.

Command:	**MOTOR:PACT,Position**<CR><LF>
Query:	**MOTOR:PACT**<CR><LF>

Arguments

Position	INT

The target position in steps.

Minimum:	-8388608
Maximum:	8388607

Returns

The absolute position, as above.

Remarks

Query is applicable any time, Set requires the motor in standby condition.

Examples

```
Tx: MOTOR:PACT<CR><LF>                              // Query
Rx: 0x0000,0x0000,1000.00<CR><LF>
Tx: MOTOR:PACT,0<CR><LF>                            // Set actual position 0
Rx: 0x0000,0x0000,0.00<CR><LF>
```

MOTOR:PREL – Relative position

Gets or sets the relative position counter in steps.

Use this function to perform relative movement, while still retaining reference to absolute position viaMOTOR:PACT. Set the desired value then use the MOTOR:RUNR command to initiate movement.

Command:	**MOTOR:PREL,Position**<CR><LF>
Query:	**MOTOR:PREL** <CR><LF>

Arguments

Position	INT

The target position in steps.

Minimum:	-8388608
Maximum:	8388607

Returns

The relative position, as above.

Remarks

Set requires the motor be in standby condition.

Examples

```
Tx: MOTOR:PREL<CR><LF>                              // Query
Rx: 0x0000,0x0000,1000.00<CR><LF>
Tx: MOTOR:PREL,0<CR><LF>                            // Set relative position 0
Rx: 0x0000,0x0000,0.00<CR><LF>
```

TZW – Zero wait time

Gets or sets the waiting time after ramping down to a stop before the next movement or direction inversion can start. Can be used to avoid excess acceleration, e.g. from MOTOR:VSTOP to MOTOR:VSTART.

When using higher values for the start and stop frequency, a subsequent move in the opposite direction would result in a jerk equal to start frequency + stop frequency. The motor may not be able to follow this. Zero wait time can be used to introduce a short delay between the two and eliminate the jerk.

Command:	**MOTOR:TZW,Duration**<CR><LF>

Query:	**MOTOR:TZW** <CR><LF>

Arguments

Duration	FLOAT

The waiting time in seconds.

[Default:	0]
Minimum:	0
Maximum:	2.7 s

Returns

The zero wait time, as above.

Examples

```
Tx: MOTOR:TZW,0.1<CR><LF>                         // Set TZW to 100 ms
Rx: 0x0000,0x0000,1.0000E+02<CR><LF>
Tx: MOTOR:TZW<CR><LF>                             // Query
Rx: 0x0000,0x0000,1.0000E+02<CR><LF>
```

MOTOR:THIGH – Microstep transition

Gets or sets the full step / microstepping transition. When frequency falls below this threshold (approximately), the motor switches from full step to the selected microstep resolution. The product determines the upper threshold automatically and applies hysteresis to avoid possible jitter between the two stepping modes. The upper threshold cannot be adjusted.

Command:	**MOTOR:THIGH, Threshold**<CR><LF>
Query:	**MOTOR:THIGH** <CR><LF>

Arguments

Threshold	FLOAT

Threshold in frequency Hz.

[Default:	10000 Hz]
Minimum:	1 Hz
Maximum:	15000 Hz

Returns

User value (data 1) and real value (data 2). See user/real values.

Remarks

AML Device control software calculates and displays the upper threshold value for reference, although as noted above it cannot be adjusted.

Examples

```
Tx: MOTOR:THIGH,500<CR><LF>                       // Set threshold to 500 Hz
Rx: 0x0000,0x0000,5.0000E+02,5.0400E+02<CR><LF>
Tx: MOTOR:THIGH<CR><LF>                           // Query
Rx: 0x0000,0x0000,5.0000E+02,5.0400E+02<CR><LF>
```

Step/Direction

MOTOR:EDGE – Edge to step on

Gets or sets which edge(s) a step occurs on when in step direction mode.

Command:	**MOTOR:EDGE, Edge**<CR><LF>
Query:	**MOTOR:EDGE** <CR><LF>

Arguments

Edge	UINT

Edge(s) to step on.

[0:	Rising edge only]
1:	Both rising and falling edges

Returns

Edge(s) to step on.

Remarks

Use option for both edges to halve the frequency on the step input required to obtain a given step rate.

Examples

```
Tx: MOTOR:EDGE,1<CR><LF>                // Set step on both edges
Rx: 0x0000,0x0000,1<CR><LF>
Tx: MOTOR:EDGE<CR><LF>                  // Query
Rx: 0x0000,0x0000,1<CR><LF>
```

MOTOR:INTERP – Step interpolation

Gets or sets a value indicating whether the step input should be interpolated to 256 microsteps. Applicable in Mode.StepDir mode only.

Command:	**INTERP, Interpolate**<CR><LF>
Query:	**INTERP**<CR><LF>

Arguments

Interpolate	BOOL

Enable interpolation of step input to 256 microsteps.

[0:	Normal; each step input will cause one step at the curr resolution]
1:	Interpolate; each step input will be interpolated to 256 microsteps.

Returns

True if interpolation mode is active, as above.

Remarks

Enabling this feature affords the benefits of high-resolution microstepping, without the drawback of very high step clock rates. Internal logic tracks the rate at which steps are supplied and smooths them out into 256 microsteps; e.g. if resolution is set to full-step and see <edge to step on> is set to rising, then each rising edge on the step input generates a series of 256 microsteps at the motor.

Examples

```
Tx: MOTOR:INTERP,1<CR><LF>                    // Enable interpolation
Rx: 0x0000,0x0000,1<CR><LF>
Tx: MOTOR:INTERP<CR><LF>                      // Query
Rx: 0x0000,0x0000,1<CR><LF>
```

Bake

BAKE:T – Bake temperature setpoint

Gets or sets the bake temperature setpoint. To run bake, select bake mode using the MODE, then start bake using the run bake command. Use stop command to end bake.

Command:	**BAKE:T,Setpoint**<CR><LF>
Query:	**BAKE:T** <CR><LF>

Arguments

Setpoint	UINT

Bake temperature setpoint.

[Default:	150 °C]
Minimum:	0 °C
Maximum:	200 °C

Returns

Bake temperature setpoint in °C, as above.

Examples

```
Tx: BAKE:T,100<CR><LF>                        // Set bake setpoint to 100 °C
Rx: 0x0000,0x0000,100<CR><LF>
Tx: BAKE:T<CR><LF>                            // Query
Rx: 0x0000,0x0000,100<CR><LF>
```

BAKE:RUN – Start bake

Start bake. Configure the bake temperature setpoint using BakeTemperature.

Command:	**BAKE:RUN**<CR><LF>

Examples

```
Tx: BAKE:RUN<CR><LF>                          // Run bake
Rx: 0x0000,0x0000<CR><LF>
```

BAKE:ELAPSED – Elapsed bake time

Gets the elapsed bake time.

Command:	**BAKE:ELAPSED**<CR><LF>

Returns

Elapsed time in format h:m:s where h is hours, m is minutes and s seconds.

Examples

```
Tx: BAKE:ELAPSED<CR><LF>
Rx: 0x0000,0x0000,2:34:12<CR><LF>             // Bake has run for 2 hours 34 minutes and 12 seconds
```

Boost

BOOST:EN – Boost enable

Gets or sets a value indicating whether the boost supply should be enabled. The boost supply steps up the input voltage from 48 V to 67 V to maximise motor dynamic performance. Enable for best performance. Regardless of this setting, the boost supply is disabled when input voltage falls below 48 V, or the boost enable jumper is not fitted.

Command:	**BOOST:EN,State**<CR><LF>
Query:	**BOOST:EN** <CR><LF>

Arguments

State	BOOL

Enable the boost circuit.

0:	Disable boost
[1:	Enable boost]

Returns

Enable state.

Examples

```
Tx: BOOST:EN,1<CR><LF>                          // Enable boost
Rx: 0x0000,0x0000,1<CR><LF>
Tx: BOOST:EN<CR><LF>                            // Query
Rx: 0x0000,0x0000,1<CR><LF>
```

Coms: Ethernet

COMS:NET:DHCP – DHCP

Gets or sets a value indicating whether DHCP is enabled. If enabled, DHCP (Dynamic Host Configuration Protocol) will be used to automatically assign network configuration, such as IP address and gateway, to the device.

Command:	**COMS:NET:DHCP,State**<CR><LF>
Query:	**COMS:NET:DHCP**<CR><LF>

Arguments

State	BOOL

DHCP enable state

0:	Disable DHCP
[1:	Enable DHCP]

Returns

Enable state	BOOL

Examples

```
Tx: COMS:NET:DHCP,1<CR><LF>                     // Enable DHCP
Rx: 0x0000,0x0000,1<CR><LF>
Tx: COMS:NET:DHCP<CR><LF>                       // Query
Rx: 0x0000,0x0000,1<CR><LF>
```

COMS:NET:GATEWAY – Gateway

Gets or sets the gateway address. When DHCP is enabled, the value read back will be the value assigned by DHCP rather than any value you might have set. Any value set however is retained, and will apply if DHCP is disabled at a later time.

Command:	**COMS:NET:GATEWAY,Address**<CR><LF>
Query:	**COMS:NET:GATEWAY**<CR><LF>

Arguments

Address	DOTTED DECIMAL

Returns

Address	DOTTED DECIMAL

Examples

```
Tx: COMS:NET:DHCP<CR><LF>                    // Query DHCP state
Rx: 0x0000,0x0000,1<CR><LF>                  // DHCP is on
Tx: COMS:NET:GATEWAY,192.168.1.1<CR><LF>     // Set the gateway
Rx: 0x0000,0x0000,10.0.96.1<CR><LF>          // DHCP is on, and has assigned the gateway so the
                                             returned value does not match what we set
```

COMS:NET:NETMASK – Subnet mask

Gets or sets the subnet mask. When DHCP is enabled, the value read back will be the value assigned by DHCP rather than any value you might have set. Any value set however is retained, and will apply if DHCP is disabled at a later time.

Command:	**COMS:NET:NETMASK,Mask**<CR><LF>
Query:	**COMS:NET:NETMASK**<CR><LF>

Arguments

Mask	DOTTED DECIMAL

Returns

Mask	DOTTED DECIMAL

Examples

```
Tx: COMS:NET:NETMASK<CR><LF>                 // Query
Rx: 0x0000,0x0000,255.255.248.0<CR><LF>
```

COMS:NET:IP – IP Address

Gets or sets the IP address. When DHCP is enabled, the value read back will be the value assigned by DHCP rather than any value you might have set. Any value set however is retained, and will apply if DHCP is disabled at a later time.

Command:	**COMS:NET:IP,Address**<CR><LF>
Query:	**COMS:NET:IP**<CR><LF>

Arguments

Address	DOTTED DECIMAL

Returns

Address	DOTTED DECIMAL

Examples

```
Tx: COMS:NET:IP<CR><LF>                            // Query the IP address
Rx: 0x0000,0x0000,10.0.97.70<CR><LF>
```

COMS:NET:IPCONF – Get network config summary

Outputs a summary of network configuration in human readable form.

Query:	**COMS:NET:IPCONF**<CR><LF>

Returns

Network config summary	See example below.

Examples

```
Tx: COMS:NET:IPCONF<CR><LF>
Rx: 0x0000,0x0000,<CR><LF>
Ethernet interface:<CR><LF>
  IPv4 Address. . . . . . . . . . . :10.0.97.70<CR><LF>
  Subnet Mask . . . . . . . . . . . :255.255.248.0<CR><LF>
  Default Gateway . . . . . . . . :10.0.96.1<CR><LF>
  DHCP State. . . . . . . . . . . . :Enabled<CR><LF>
```

COMS:NET:LINK – Get link up status

Gets a value indicating whether the ethernet interface link is up. This will read back as false when the LAN connector is unplugged for example.

Query:	**COMS:NET:LINK**<CR><LF>

Returns

Link up state	BOOL

Examples

```
Tx: COMS:NET:LINK<CR><LF>                          // Query
Rx: 0x0000,0x0000,1<CR><LF>                        // Link is up
```

COMS:NET:MAC – Get MAC address

Gets the Ethernet interface MAC address.

Query:	**COMS:NET:MAC**<CR><LF>

Returns

MAC address	MAC

Examples

```
Tx: COMS:NET:MAC<CR><LF>                           // Query
Rx: 0x0000,0x0000,44:b7:d0:c7:16:75<CR><LF>
```

Coms: Serial

COMS:SERIAL:BAUD – Baud rate

Gets or sets the baud rate.

Command:	**COMS:SERIAL:BAUD,Baud**<CR><LF>
Query:	**COMS:SERIAL:BAUD**<CR><LF>

Arguments

Baud	UINT

Baud rate

4800
9600
14400
19200
38400
57600
[115200]
230400
460800
921600

Returns

Baud rate	UINT

Examples

```
Tx: COMS:SERIAL:BAUD,9600<CR><LF>          // Set 9600 baud
Rx: 0x0000,0x0000,9600<CR><LF>
Tx: COMS:SERIAL:BAUD<CR><LF>               // Query
Rx: 0x0000,0x0000,9600<CR><LF>
```

COMS:SERIAL:MODE – RS232/RS485 mode selection

Gets or sets the serial coms mode, either RS232 or RS485. Unplug from the host device before changing the mode.

Command:	**COMS:SERIAL:MODE,Mode**<CR><LF>
Query:	**COMS:SERIAL:MODE**<CR><LF>

Arguments

Mode	ENUM

Serial interface mode

0:	RS232
[1:	RS485]

Returns

Mode	ENUM

Examples

```
Tx: COMS:SERIAL:MODE,1<CR><LF>             // Set RS485 mode
Rx: 0x0000,0x0000,1<CR><LF>
Tx: COMS:SERIAL:MODE<CR><LF>               // Query
Rx: 0x0000,0x0000,1<CR><LF>
```

COMS:SERIAL:RS485DEL – Turnaround delay

Gets or sets a value in milliseconds specifying the delay to execute between receipt of a command from the host and the client (SMD4) sending the response. Applicable to RS485 mode only.

The RS485 interface is half duplex (it can send or receive data, but cannot do both at once) and so by default is in the receive state. The interface switches to transmit mode when a command has been received, executed and a response is ready to send. The turnaround delay is used to insert an additional delay following execution of the command but preceding switching to transmit, to allow the host more time to switch into receive mode.

Experiment with increasing this setting if you find that host receives a response with a portion missing from the start of the response, for example missing some or all of the status an error flags.

Command:	**COMS:SERIAL:RS485DEL,Delay**<CR><LF>
Query:	**COMS:SERIAL:RS485DEL**<CR><LF>

Arguments

Delay	UINT

Delay in ms.

Default:	0
Minimum:	0
Maximum:	1000

Returns

Delay in ms	UINT

Examples

```
Tx: COMS:SERIAL:RS485DEL,10<CR><LF>          // Set delay of 10 ms
Rx: 0x0000,0x0000,1<CR><LF>
Tx: COMS:SERIAL:RS485DEL<CR><LF>             // Query
Rx: 0x0000,0x0000,10<CR><LF>
```

COMS:SERIAL:TERM – Termination

Gets or sets a value indicating whether RS485 line termination should be used. If enabled, a 120 termination resistance is placed between the RS485 A and B pins. See <section on termination>.

Command:	**COMS:SERIAL:TERM,State**<CR><LF>
Query:	**COMS:SERIAL:TERM**<CR><LF>

Arguments

State	BOOL

Termination enable state

[0:	Disabled]
1:	Enabled

Returns

Enable state	BOOL

Examples

```
Tx: COMS:SERIAL:TERM,0<CR><LF>               // Disable termination
Rx: 0x0000,0x0000,0<CR><LF>
Tx: COMS:SERIAL:TERM<CR><LF>                 // Query
Rx: 0x0000,0x0000,0<CR><LF>
```

COMS:SERIAL:SLAVEADDR – Slave address

Gets or sets the slave address. Only applicable when addressing mode is used, see <section at top here detailing addressing>

Command:	**COMS:SERIAL:SLAVEADDR,Address**<CR><LF>

Query:	**COMS:SERIAL:SLAVEADDR**<CR><LF>

Arguments

Address	UINT

Termination enable state

Default:	1
Minimum:	1
Maximum:	247

Returns

Address	UINT

Examples

```
Tx: COMS:SERIAL:SLAVEADDR,1<CR><LF>                    // Disable termination
Rx: 0x0000,0x0000,1<CR><LF>
Tx: COMS:SERIAL:SLAVEADDR<CR><LF>                      // Query
Rx: 0x0000,0x0000,1<CR><LF>
```

Guidance on use of VCSMs

It is assumed that the reader is familiar with the production and handling of UHV components. The successful application of vacuum stepper motors requires an appreciation of their thermal as well as their mechanical properties. Compared to motors operated in air, the available cooling means for motors in vacuum are much less effective.

Apart from extending the run time, operation at a low temperature improves the outgassing performance of motors. Therefore, minimum running times and motor currents should always be pursued. Selection of the largest motor possible for the application will result in longer running times, lower motor temperature and lowest outgassing.

Design mechanisms with balanced loads whenever possible or arrange that either the static friction in the system or the motor detent torque will hold position without the necessity of maintaining phase currents to produce a holding torque. The IH command may be used to reduce the phase currents and produce a holding torque which is intermediate between the pull-out torque and the detent torque. Refer to section Low power techniques for a full description of power reduction techniques.

Many applications that appear to require continuous running, for example, substrate rotation for ensuring uniformity of deposition or implantation, can be equally well performed by intermittent short periods of stepping at low duty cycle. Stepper motors should not be disassembled as this partially demagnetises the permanent magnet in the rotor and permanently reduces the torque.

Operating temperature and run times

The maximum recommended running temperature of AML motors is 190 °C, as measured by the embedded type K thermocouple or RTD.

Current D-series motors have published temperature and time graphs for typical operating conditions with the motor mounted by its flange. Continuous running can readily be achieved with care at medium phase currents. Run times at higher currents can be increased by additional heatsinking at the other end of the motor.

Some AML motors are suitable for operation at 77 °K and they are believed to be suitable for use at lower temperatures. Because the resistance of the windings at low temperatures is small, the efficiency of the motor is much greater than at normal temperatures. A resistance of a few ohms should be connected in series with each winding, in order to present a normal load to the SMD4. The leads of the motor will be very brittle at low temperatures and should not be allowed to flex. The normal mechanical and electrical properties of all materials are recovered on return to room temperature.

Outgassing and bakeout

Newly installed motors will outgas, mainly due to water-vapour retention in polyimide. As this material is microporous the water is released rapidly, and the rate will subside after a few hours. The rate may be accelerated by running the motor to self-heat it.

Baking at up to 200 °C is permissible, and a 24-hour bake at this temperature will normally reduce the outgassing to its minimum.

Motors are typically operated at some distance from the chamber walls where the bakeout temperature is most often controlled. If the temperature indicated by the motor temperature sensor during bakeout is not high enough when the bakeout period is well advanced, it may be increased to 200 °C by using the bake mode. This energises both phases, keeping the motor stationary in a half-step position. Phase current is modulated to achieve the programmed setpoint. Keeping the motor hot by this means while the rest of the vacuum system cools is recommended as this will prevent condensation on the motor.

Where internal infra-red heaters are used for bakeout it is advisable to shield the motor from direct radiation and to achieve the desired temperature during bakeout by using the bake program.

Irreversible deterioration of the winding insulation will begin to occur above 230 °C and the motor may subsequently produce larger amounts of gas, even at lower temperatures.

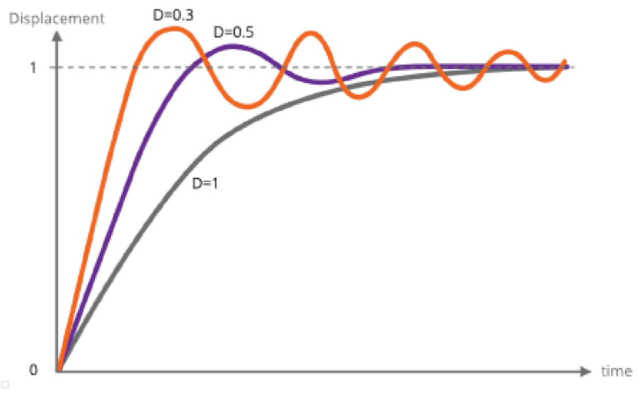
Resonances

Stepper motors are classic second-order systems and have one or more natural resonant frequencies. These are normally in the 50 – 100 Hz region for unloaded motors. Operation at step rates around these frequencies will excite the resonances, resulting in very low output torques and erratic stepping. Another set of resonances can occur in the 1 – 2 kHz region, but these do not normally present any practical problems.

Load inertia, friction and drive characteristics

The primary (lower) resonant frequency cannot be stated with any precision, since it is modified by the friction and inertia of the load, the temperature of the motor and by the characteristics of the drive. Coupling a load inertia reduces the resonant frequency and decreases the damping factor. Load friction increases damping. Because the drive circuits of the SMD4 produce a controlled phase current this produces heavy damping. Drives which are voltage sources and which rely on the motor winding and other resistance to define the current have a lower damping factor.

The effect of changing the damping on the single step response of the motor is shown in the diagram below.



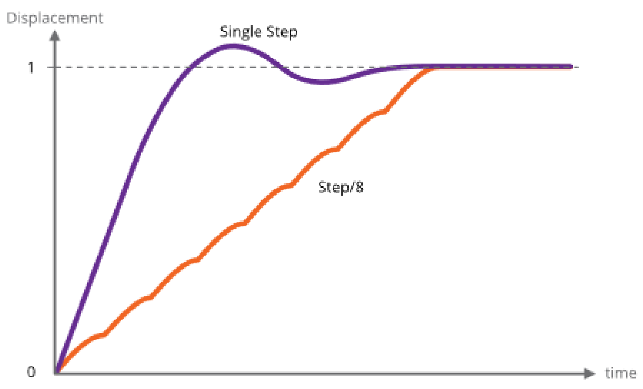
Control of resonance

The simplest method of controlling resonances is to avoid operation of the motor close to the resonant frequencies. It is usually possible to start a motor at rates in excess of 300 Hz if the load inertia is small, thereby completely avoiding the primary resonance. Resonances are not usually a problem when the motor speed is accelerating or retarding through the resonance frequency region.

If it is necessary to operate at slow speeds or with large load inertia, using microstepping helps. It effectively increases the stepping rate by the step division factor and reduces the amplitude of the step transients that excite the resonances. This is shown in the diagram below. Because both phases are energised in microstepping there are some other processes of interchange of energy between the windings which do not occur in the single step mode and these increase the damping factor.

In particularly difficult cases, modifying the step frequency at which transition from full stepping to microstepping occurs can be helpful.

A typical motor response to a single step and to a single step subdivided into eight microsteps is shown in the diagram below.



Mechanisms for use with VCSMs

The following section is an introduction to this topic and is intended to indicate the major mechanical and vacuum considerations for various types of mechanisms. A working knowledge of mechanics and vacuum construction techniques is assumed. AML supply a range of standard mechanisms which can be customised, as well as designing custom mechanisms and components.

Rotation (Position control)

The load inertia coupled to the motor shaft should ideally be small compared to the rotor inertia of the motor. Load inertia up to two or three times that of the motor can be driven, without significant difference to the maximum start speed and acceleration which is achieved by the unloaded motor. Load inertia of around ten times that of the motor can be driven with absolute synchronism, provided care is taken over specifying the microstep and acceleration parameters. Larger inertia loads should be driven through reduction gearing.

Significant loads should have their centre of gravity on their axis of rotation, unless they are rotating in a horizontal plane.

Angular resolution at the motor shaft is limited to a single step of 1.8 °. The actual rest position within the step is determined mainly by the load friction and any torque imposed by the load on the motor at rest. If the rotor position is displaced θ° from the nominal step position, the restoring torque increases approximately in proportion to sin(100 × θ)° The maximum torque at the half step position is either the detent torque or the holding torque, depending on whether the motor is powered at rest. If the static friction and any torque due to an unbalanced load are known, this allows the rest position error to be estimated using the above approximation. The friction within the motor bearings is very low, so that a completely unloaded D42.2 motor will normally settle within 0.2 ° of the desired position if brought suddenly to rest from full stepping at 300 Hz.

Angular resolution may be improved by reduction gearing: this is discussed below.

Rotation (Speed control)

In some applications, the precise position of a rotating load is not important or can be deduced by other means, but the speed of rotation may need to be controlled very precisely. Beam choppers and sample rotators for control of deposition uniformity are applications of this type. An increased load inertia may be desirable to smooth out the stepping action of the motor. Loads of up to about 1000 times the inertia of the motor can be controlled by using long acceleration ramps. Some steps may be lost during acceleration and retardation of such loads, but precise synchronism at constant stepping frequency is easily achieved and recognised.

Significant rotating loads should be balanced, at least to the extent that the torque presented to the motor shaft is less than the detent torque of the motor. The motor torque requirement will then be dominated by that required to accelerate the load.

A typical example of a large inertia load was a 1.5 kg disk of uniform section, 20 cm in diameter. This was directly coupled to a D42.2 motor and rotated continuously in vacuum at 30 RPM.

Translation

Translation may be produced by a leadscrew and nut, wire-and-drum or rack-and-pinion mechanisms. The choice depends on the precision, length of travel, force and speed required. Leadscrew-based translators are capable of exerting forces of kilograms with resolutions of a few microns per step.

Accurate leadscrews are practical up to 400 mm long. With anti-backlash gearing between the motor and leadscrew resolution of one micron is practical. Anti-backlash nuts are not normally necessary for vertical motions. If a conventional nut is used with the leadscrew the load will be dominated by friction, especially if there is a reduction gear between the lead screw and the motor shaft which reduces the reflected load inertia.

Because of the lubrication restrictions and the slow speeds of UHV mechanisms the static friction is usually much more significant than dynamic friction. The optimum material for nuts is phosphor bronze and for lead screws is stainless steel with a diamond-like coating (DLC). DLC has a very low coefficient of friction in vacuum. Burnishing or sputtering a layer of pure Molybdenum Disulphide on the leadscrew may be useful in reducing friction and wear. The typical coefficient of friction between these materials is 0.1 and typical efficiencies are 40 % with ground trapezoidal threads. The gas load generated by frictional heating of the leadscrew is usually somewhat less than that of the motor.

The frictional losses in drum or rack drives are lower than in conventional leadscrew drives and considerations of inertia usually dominate. Rack and pinion drives are suitable for travel up to a few hundred millimetres and wire and drum mechanisms may be made several metres long. Another alternative for heavy loads is a studded stainless-steel band and matching pulleys. The repeatability and backlash of all these alternative translation drives are much worse than with screw-driven schemes.

Linear guides

Low-cost translation mechanisms can use simple bushes running on ground stainless-steel rods. A variety of carbon-reinforced polymer materials, such as PEEK, are suitable for the bushes.

'V' groove rollers and tracks and crossed-roller guides are suitable for more accurate translators. The former have the advantage of being practical to 1 metre and have minimal overall length for a given travel. Crossed-roller slides are more rigid and can support larger loads, but at higher cost. Both types have preload adjustments. 'V' rollers have smaller load-bearing surfaces and only have a rolling contact at a single point and are consequently liable to greater wear if heavily loaded. AML products of the VSM23 and VSM17 series are small-dimension examples of these types of mechanisms.

Reduction gearing

The inertia of loads coupled by reduction gearing is reduced at the motor in proportion to the square of the reduction ratio. Where reduction gearing is used for load matching, the spur gear meshing with the motor pinion will normally dominate the load inertia and it is important to keep its diameter small. Anti-backlash gears and standard pinions should be used in the gear train to damp any resonances in the mechanism. Gears for use in UHV should be designed for low friction without lubrication and with dissimilar materials in contact to avoid cold-welding. Nitrogen ion-implantation of the rolling surfaces or complete Titanium Nitride coating of gears are effective means of achieving this and other desirable properties in all-stainless-steel gear trains.

Bearings

Bearings for use in UHV should be unshielded and have a stainless steel cage and race. The balls should be either stainless steel coated with some other material or solid ceramic. As an alternative, all-stainless bearings having a PTFE composite component in the race (which is designed to transfer to the balls) are also suitable.

Magnetic fields near motor

Motors should not be operated in fields of greater than 50 millitesla (500 gauss), as this will affect the performance while the field is present. Fields significantly greater than this may cause partial demagnetisation of the rotor, reducing the torque. Demagnetised motors can be restored by AML.

The leakage field of a motor is of the order of 1 millitesla (10 gauss) at 1 cm from the cylindrical surface of the motor in an axial direction and is present when the motor is not powered. Under drive an alternating component is added at the step frequency and its harmonics up to a few kHz. The field is easy to screen with Mu-metal or similar high permeability foil to below a few milligauss at the sides of the motor but is more difficult around the projection of the shaft. Early consideration of the interaction of stray fields on nearby equipment is recommended.

Low power techniques

In the design of small mechanisms there are several factors that are not accurately known, or that have poor tolerances, for which generous allowances must be made. The result should be a conservative design where the available torque is in excess of the requirement. Some of this excess can be exchanged for increased running time or decreased outgassing in vacuum by various techniques. Used in combination the improvement can be very significant.

Most of the 'tuning' procedures below require the motor or mechanism to be run on the bench under realistic representative operating conditions while adjusting a parameter to the point where normal stepping operation fails. Erratic stepping is easy to see; a cable tie on the motor shaft makes a useful pointer. Familiarity with the SMD4 software and or remote interface is assumed.

Techniques applicable to all applications

The following techniques can be used in all applications to reduce motor power:

1. Run the motor at stepping rates between 500 Hz and 2 kHz, where its electromechanical efficiency is greatest, if possible
2. Reduce the acceleration for inertia-dominated loads
3. Reduce the phase current progressively to about 20% more than the minimum for consistent stepping. Some adjustment of the acceleration parameters may be needed
4. Make use of the run and acceleration current settings; try a higher current during acceleration to overcome inertia of a large load, and the minimum current possible during run to keep the load moving. This will reduce

motor power dissipation versus using the same higher current all the time

5. Improve the heatsinking arrangements. A reduction in motor temperature decreases the winding resistance and increases its efficiency.

Techniques where step rates less than 100 Hz

For applications where operation below few hundred steps per second is satisfactory, use the following technique beyond those above. The desired effect is to complete each step as quickly as possible and remove or reduce the power to the minimum as soon as possible. Each step is completed in a few milliseconds, so that the power saving is progressively greater at lower speeds. Read and ensure that you understand the complete procedure before starting.

Set the target frequency VMAX equal to the start speed VSTART and steadily increase them both to determine the highest speed at which the motor will start. Take care when increasing the speed through the expected resonance range because the motor may not start in that range, although it may start reliably at higher speeds.

Configure PDDEL, IHD and IH to zero, to reduce motor current to zero as quickly as possible after each step is completed. Move the motor in single steps at the highest reliable starting speed, followed by a delay of a few milliseconds or more. This reduces power dissipation in the motor and so minimises temperature rise. It also has the consequence of reducing the damping factor at the time the power is reduced, so some experimentation with the parameters is required to ensure an adequate margin of stability is obtained.

Possible causes of damage to VCSMs

Vacuum motors must be de-magnetised before disassembly and re-magnetised and cleaned after repair. For these reasons most will need to be returned to AML for repair. The notes below offer guidance on the avoidance of the most common problems and diagnostic advice.

Bearing damage

The ceramic balls in the bearings are very strong but more brittle than steel balls. Dropping the motor on its end will probably break some balls. The damage is occasionally visible and any roughness felt when rotating the shaft manually will indicate that this has happened.

Debris inside the motor

Foreign material can enter the motor via the pumping holes and gaps in the bearings. Particles of magnetic materials are particularly likely to be attracted through the pumping holes and they eventually migrate into the gap between the rotor and stator. They usually cause the rotor to stick at one or more points per revolution and can often only be felt when rotating in a specific direction. Fortunately, the larger motors have enough torque to grind them into a dust.

The main cause of this type of problem has been users modifying shafts. This can be avoided by sealing the motor inside a cleaned polyethylene bag and supervising the machining closely. Clean the projecting shaft and remove magnetic particles with a magnet before opening the bag. Remove the motor or similarly seal it if any filing or drilling of nearby components is done.

Overheating

Motors which have been heated to 230 °C will produce a much greater gas load thereafter, although their electromechanical performance may not be affected. In extreme cases, the insulating material will ablate and deposit itself as a yellow powder inside the motor case and on any cool surfaces in line with the pumping holes.

Motors can overheat very quickly in vacuum. This is very unlikely to happen with a properly connected SMD4 drive. Never use a drive capable of providing more than 1 amp of phase current and ensure that the drive current is removed as soon as the indicated temperature exceeds 190 °C. This is performed automatically by the SMD4.

Maintenance and service

The SMD4 contains no user-serviceable parts.

Cleaning

If the instrument requires cleaning, disconnect the power and all other connections, and wipe it down with a cloth slightly dampened with water or a mild detergent.

Troubleshooting

Problem	Resolution
Red and green status indicators off	Check that the power supply to SMD4 is correctly connected and meets the requirements given in section Technical Information
Red status indicator flashing or lit solidly	Each indication corresponds to a fault, review section Faults
Motor does not move when commanded	Is a fault present? Check the red status indicator on the front panel, and use the SMD3 software or remote interface to determine the nature of the fault. After the fault has been fixed, run the CLR command, or use the reset fault input to restore normal operation Check the motor wiring to the SMD3; disconnect the motor, and use a multimeter to measure the resistance of each phase, see section Lead Identification, which should be approximately 3 – 15 ohms
Joystick does not work	Is joystick mode selected? If you want to switch to joystick mode automatically on connection of the joystick, see setting AUTOJS and section Joystick. The joystick must be wired correctly for the auto-detect function to work as described in section Joystick.

Storage and disposal

The product must be disposed of in accordance with the relevant local regulations for the environmentally safe disposal of systems and electronical components.

In the United Kingdom (UK) and European Union (EU), waste from electrical and electronic equipment (WEEE) is subject to legislation designed to prevent the disposal of such waste and to encourage proper treatment measures to minimize the amount of waste ultimately disposed to landfill. To view AML’s WEEE policy please visit: https://arunmicro.com/documentation/WEEE_procedure.pd

Assistance

In the first instance, contact the distributor or supplier of the equipment. Always quote the serial number of the instrument and firmware and software versions. Provide a written description of the problem. If the problem is related to a motor or mechanism manufactured by AML, include the serial number(s) of those items. Do not return products to AML without prior approval.

Arun Microelectronics Ltd
Tel: +44 (0)1903 884141
Email: info@arunmicro.com
Website: arunmicro.com

Compliance Certificate

This declaration of conformity is issued under the sole responsibility of the manufacturer.

Manufacturer:	Arun Microelectronics Limited
Address:	Unit 2, Bury Mill Farm, Bury Gate, Pulborough, RH20 1NN, United Kingdom
Object:	Stepper Motor Drive
Part No.:	SMD4

The object of the declaration described above is in conformity with the relevant UK Statutory Instruments (and their amendments), and the relevant European Union harmonisation legislation:

Statutory Instruments:	2016 No. 1091	The Electromagnetic Compatibility Regulations 2016
	2016 No. 1101	The Electrical Equipment (Safety) Regulations 2016
	2012 No. 3032	The Restriction of Use of Certain Hazardous Substances in Electrical and Electronic Equipment Regulations 2012
Directives:	2014/30/EU	EMC Directive
	2014/35/EU	Low Voltage Directive
	2015/863	RoHS Directive
Standards:	Harmonised and international/national standards and specifications:	
	EN 61010-1:2010+A1:2019	Safety requirements for electrical equipment for measurement, control and laboratory use
	EN IEC 61800-3:2023	Adjustable speed electrical power drive systems

Signature [illegible]

Mr. R Burling, Product Design Manager

Place, Date Pulborough, April 2024