Software

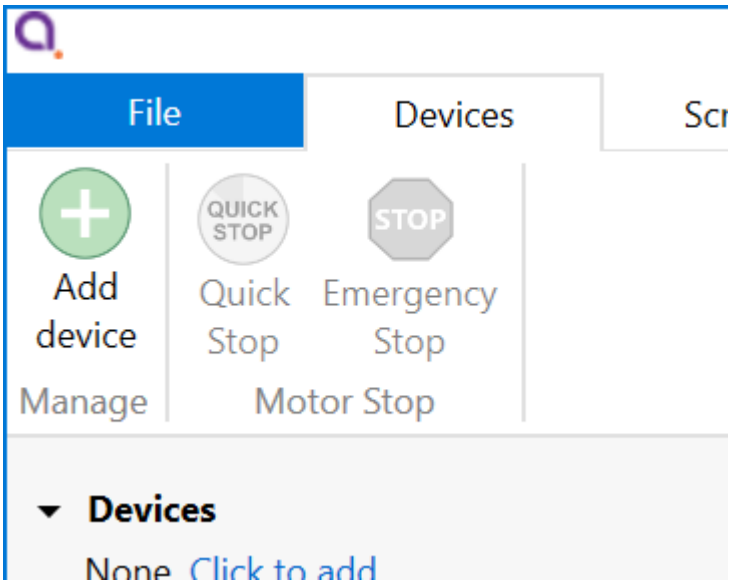
Installation and setup

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

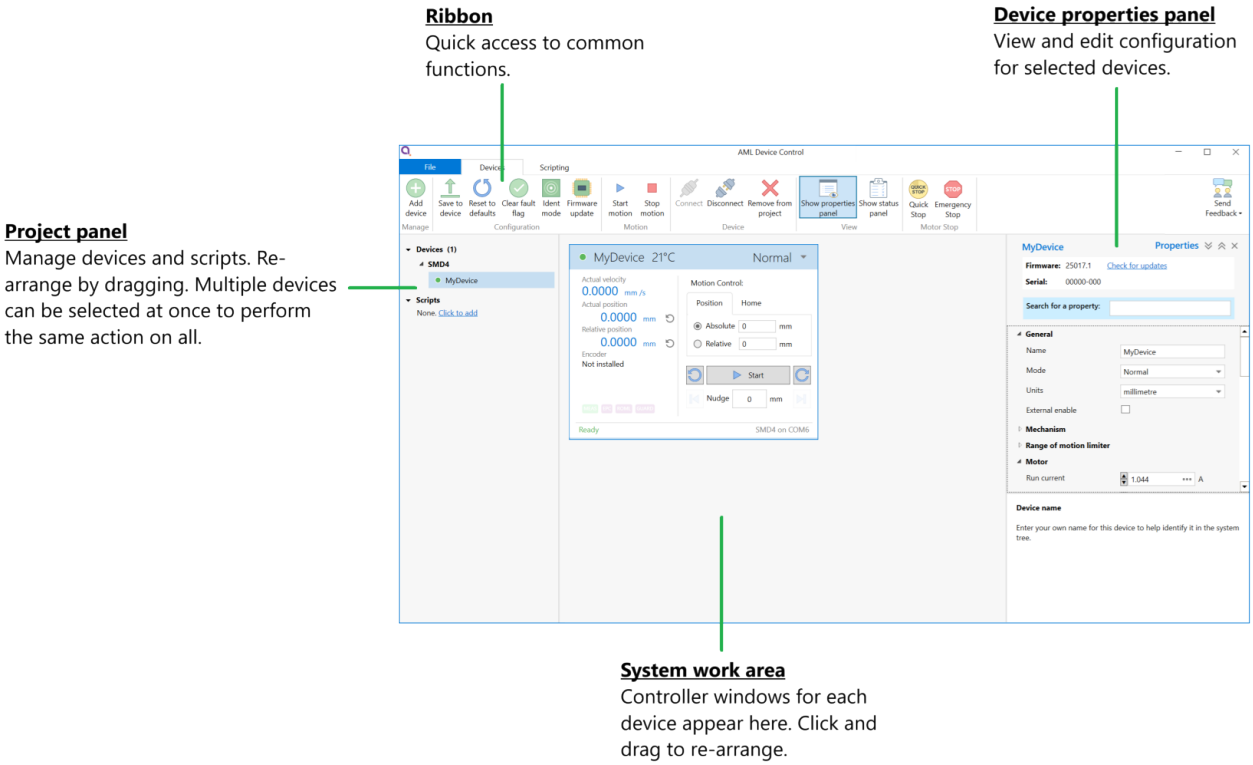
Cancel

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 its IP address.

Overview

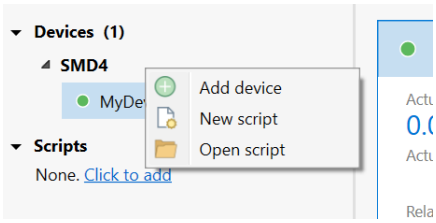
The default layout of the software is shown below:



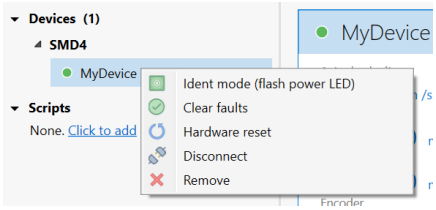
Project panel

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.



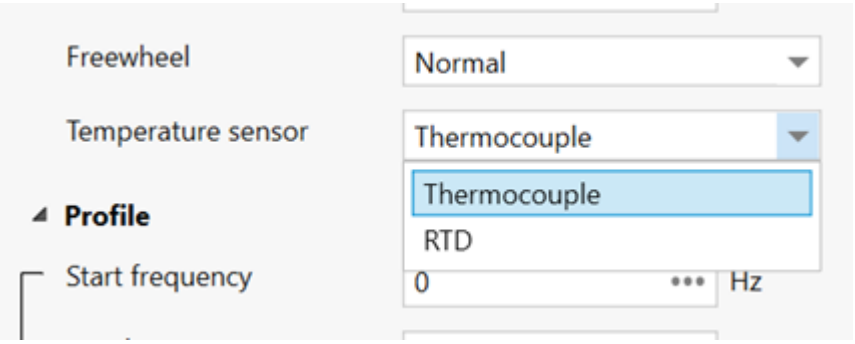
A status indicator next to each device shows the current status of each device according to these colours:

Colour	Description
	Device connected and ready
	stick connected
	Device in a fault state
	Device disconnected

Device properties panel

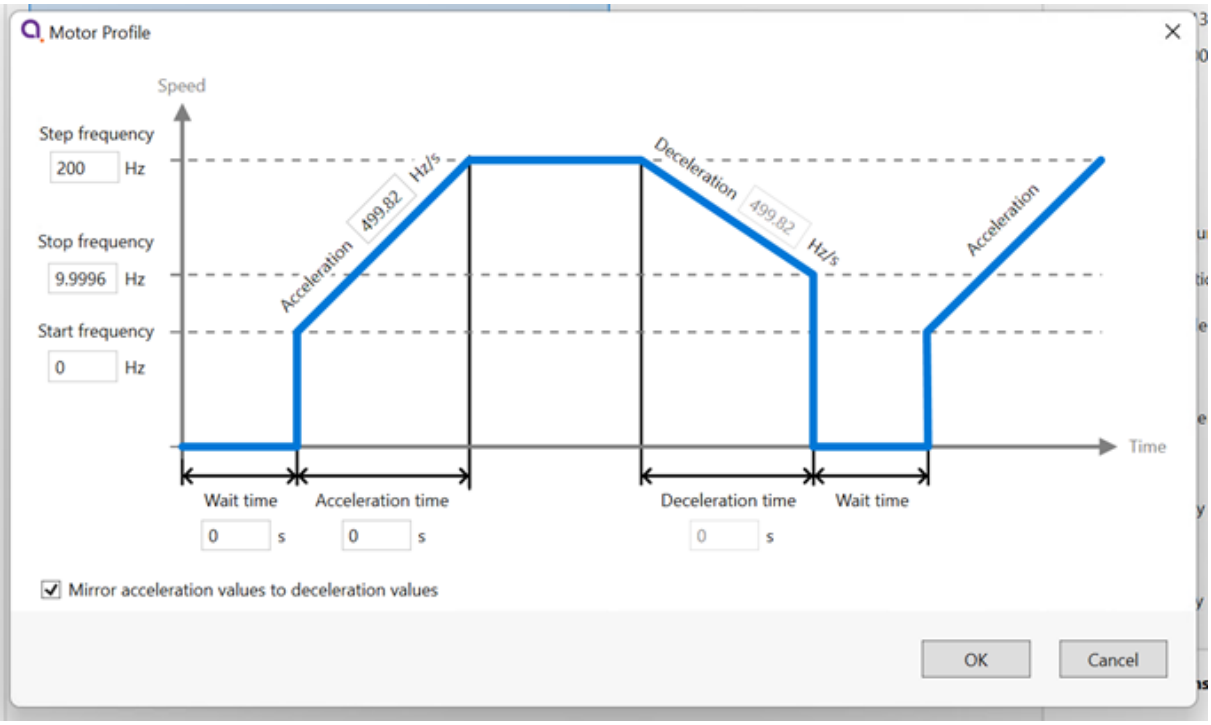
Select one or more devices in the project panel; their properties are displayed and can be edited here. A blank is shown for properties which are different across the selected devices. As each property is selected, help text appears at the bottom of the panel describing the configuration option in more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

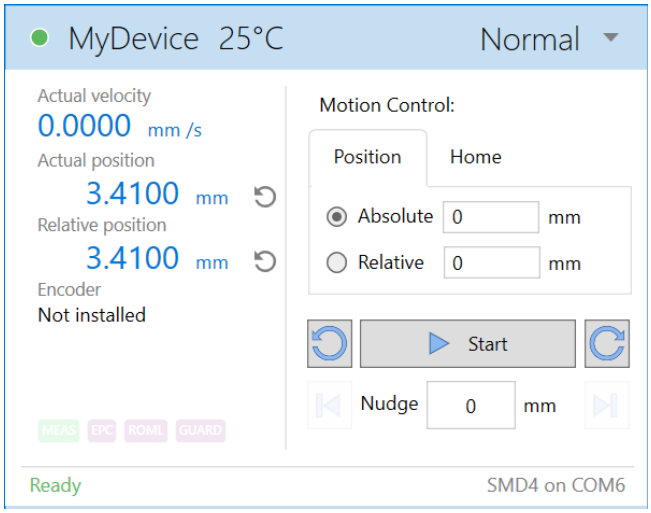
Controller windows for each device appear in this area. Windows can be arranged as desired and will automatically ‘snap’ to a grid making it easy to keep them neatly organised.

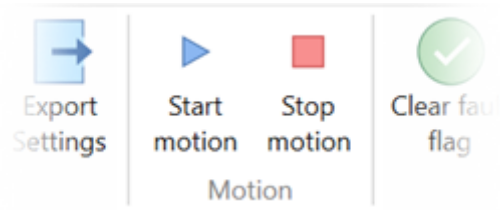
Controller window

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 may be reset using the ↺ icons adjacent to the readouts.

Individual devices may be controlled using the buttons on the controller window, or if multiple devices are selected in the system tree on the left, use the controls on the ribbon to operate them all at once.

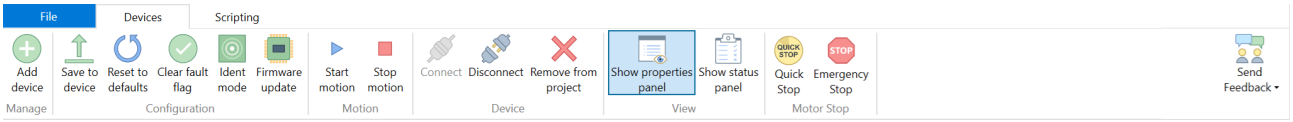




> Be aware that the synchronisation between multiple SMD4s is not specified or guaranteed. For example, if multiple devices are selected and the start button on the ribbon is clicked delays within the computer, software, and data connection to the SMD4 mean that each SMD4 will start or stop its motion at a slightly different moment, therefore this option is not suitable for performing complex co-ordinated movements across multiple axes.

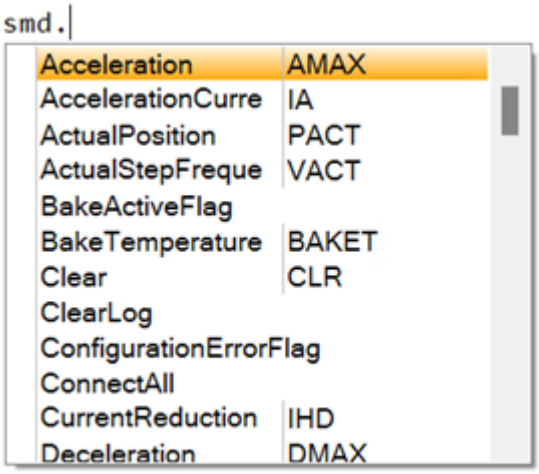
Ribbon

Contains buttons for all actions. Hover over a button with the mouse to show tooltip with help information.



Scripting

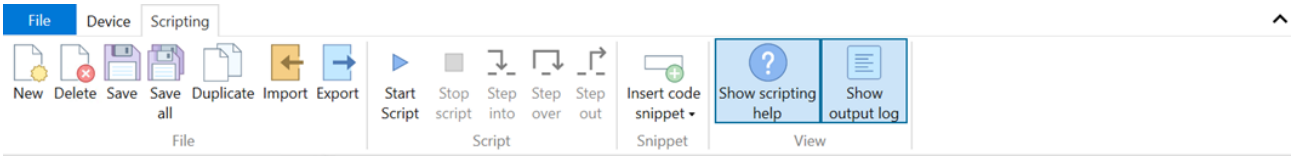
The software includes an easy to use script editor, that allows for sequences to be programmed and executed on multiple connected SMD4 devices, as well as system level operations such as adding and removing SMD4 devices from the project.The scripting language used is JavaScript; this is powerful, easy to use and extensively documented. A global ‘smd’ object is made available from which you perform all interactions with the SMD4s. Type ‘smd.’ (or use the 'CTRL+K' keyboard shortcut) 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.



Information on the available SMD4 device specific commands can be found in section USB 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(1);                    // Set the SMD4 mode to 1 (Normal)
smd.Mode();                     // Query state of mode
```

The ribbon contains scripting specific buttons.



Utility functions

These utility functions don't translate to a command executed by the SMD4, and are used to perform tasks such as connecting devices, or creating delays in your program.

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 for a specified duration in milliseconds, before executing the next instruction. value: Minimum: 0 Maximum: 2^31 -1
DelaySeconds	void **DelaySeconds**(int **value**) Delay for a specified duration in seconds, before executing the next instruction. 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);
```

Execute a series of movements, illustrating synchronous move commands

```
// Select device and set mode to remote
smd.Select("MyDevice");
smd.Mode(1);
// Move +100 steps, then -500 steps, and finally +100 steps
// "MoveRelative" executes synchronously, i.e. the motor must arrive at its destination position before the next line of code executes
smd.MoveRelative(500);
smd.MoveRelative(-500);
smd.MoveRelative(100);
// Log the finishing step position to the command line output area
smd.Log(smd.ActualPosition()[0]);
```

Execute movement on multiple drives, illustrating synchronous and asynchronous commands

```
// Select first device and start a movement
smd.Select("Device_0");
smd.Mode(1);
smd.MoveRelativeAsync(10000); // Execution continues to the next line of code immediately, regardless of how long this move will
                              // requires to complete.
// Select second device and start a movement
smd.Select("Device_1");
smd.Mode(2);
smd.MoveRelativeAsync(5000);

// Both motors are completing their moves at the same time.
```

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");
}
```