



User Manual

Motion86

86Duino Coding IDE 501+
Motion86 Library

(Version1.0)

REVISION

DATE	VERSION	DESCRIPTION
2026/9/9	Version1.0	New Release.

COPYRIGHT

The information in this manual is subject to change without notice for continuous improvement in the product. All rights are reserved. The manufacturer assumes no responsibility for any inaccuracies that may be contained in this document and makes no commitment to update or to keep current the information contained in this manual.

No part of this manual may be reproduced, copied, translated or transmitted, in whole or in part, in any form or by any means without the prior written permission of the ICOP Technology Inc.

©Copyright 2026 ICOP Technology Inc.

Ver.1.0 September, 2026

TRADEMARKS ACKNOWLEDGMENT

ICOP® is the registered trademark of ICOP Corporation. Other brand names or product names appearing in this document are the properties and registered trademarks of their respective owners. All names mentioned herewith are served for identification purpose only.

For more detailed information or if you are interested in other ICOP products, please visit our official websites at:

- Global: www.icop.com.tw
- USA: www.icoptech.com
- Japan: www.icop.co.jp
- Europe: www.icoptech.eu
- China: www.icop.com.cn

For technical support or drivers download, please visit our websites at:

- https://www.icop.com.tw/resource_entrance

For EtherCAT solution service, support or tutorials, 86Duino Coding IDE 500+ introduction, functions, languages, libraries, etc. Please visit the QEC website:

- QEC: <https://www.qec.tw/>

Content

Content	iii
Ch. 1 Introduction.....	1
1.1 About Motion86 Library	2
1.1.1 Key Features	2
1.1.2 Architectural Evolution	2
1.2 What is 86Duino IDE?.....	3
1.3 Machine Types	4
1.4 Hardware Wiring.....	5
1.4.1 Local Machine Pin Assignment	5
1.4.2 Wiring Examples	6
1.4.3 EtherCAT Servo Configuration	8
1.5 Motion Methods	9
1.5.1 Linear Motion	10
1.5.2 Arc Motion	11
1.5.3 Circle Motion	14
1.5.4 Helical (Spiral) Motion	15
1.6 Architecture	16
1.7 Functions Table	20
1.7.1 Machine Class.....	20
1.7.2 EcatMachine Class	24
1.7.3 Custom Machine Tools	25
Ch. 2 Functions.....	26
2.1 Machine Class	27
2.1.1 Basic Method	28
2.1.2 Basic Control	45
2.1.3 Write Configuration.....	72
2.1.4 Read Configuration	101
2.1.5 JOG / MPG Control	123
2.1.6 Feedrate Override.....	148
2.1.7 Axis Following (Electronic Gear)	155
2.1.8 Calculation Tools	160
2.2 EcatMachine Class	164
2.2.1 Constructor / Destructor	165
2.2.2 Attach EcatServo Method	169
2.2.3 Attach Home Pin Method.....	174
2.2.4 Attach Jog Pins Method.....	179
2.2.5 Attach Mpg Method	184
2.2.6 Basic Method (overridden).....	187

2.3	Custom Machine Tools	190
2.3.1	Callback Functions	191
2.3.2	Plugin Override Points	200
Ch. 3	Example.....	202
3.1	Local Machine	203
3.1.1	Example: Basic Move	203
3.1.2	Example: Basic Move with G-code	205
3.1.3	Example: JOG Control	207
3.1.4	Example: MPG Control	209
3.1.5	Example: Read G-code File.....	211
3.1.6	Example: Feedrate Override	214
3.1.7	Example: Axis Following	218
3.1.8	Example: Multiple Machines	222
3.2	EcatMachine	227
3.2.1	Example: Basic Move with EtherCAT Servos	227
3.2.2	Example: EtherCAT Homing, Jog and Hand Wheel.....	229
3.3	Custom Machine	231
3.3.1	Example: Custom Machine with Callbacks	231
Appendix		233
A.1	G-code Command Reference.....	234
A.2	Enumerations	239
A.3	Usage Restrictions.....	240
Warranty		241



Ch. 1

Introduction

1.1 About Motion86 Library

86Duino IDE has added this library since Coding version 318, enabling 86Duino to become a simple multi-axis motion controller.

Motion control is a part of automation technology. Industrial automation can increase production speed, reduce production cost, improve production quality, and enhance market competitiveness, so motion control is often used in packaging, printing, textile, semiconductor device manufacturing, and Production lines.

1.1.1 Key Features

The **Motion86** library enables the 86Duino to become an easy-to-develop motion controller with Arduino-friendly features for automation applications, speeding up controller development and reducing development costs.

The main features of the Motion86 library are as follows:

- Up to 9 axes of stepper and servo motor motion control with up to 10MHz pulse output.
- Up to 3 independent axis groups.
- G-Code support.
- Multi-axis linear, circular, and spiral interpolation with 0.5ms fine interpolation cycle.
- Support JOG function.
- Support handwheel (MPG).
- Speed Look Ahead program pre-reads 1024 sessions.
- Feedrate Override for immediate response.
- Shaft follows function with electronic gear ratio setting.

1.1.2 Architectural Evolution

Starting from 86Duino Coding IDE 501, the **Motion86** library has undergone a significant architectural update. While maintaining support for local I/O configurations (**Machine**), the library now introduces an EtherCAT-based multi-axis motion controller (**EcatMachine**).

This evolution significantly enhances system scalability and decouples the Motion86 library from specific hardware constraints, enabling a more flexible and robust motion control solution.

Additionally, the new **CustomMachine** feature has been introduced. This allows developers to seamlessly integrate devices utilizing various communication protocols, facilitating highly customized motion control implementations tailored to specific industrial needs.

1.2 What is 86Duino IDE?

The 86Duino integrated development environment (IDE) software makes it easy to write and upload code to 86Duino boards and QEC MDevice. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software, which can be downloaded from <https://www.qec.tw/software/>.



For more information, please visit [86Duino](https://www.qec.tw/) Website.

1.3 Machine Types

A `Machine` object controls one **axis group** of up to three axes (X, Y, Z). Motion86 distinguishes two kinds of axis groups, selected by the `axisGroup_t` value passed to the constructor:

Kind	axisGroup_t	Actuator	Position output
Local Machine	MACHINE_0, MACHINE_1, MACHINE_2 (0~2)	Stepper or pulse-input servo drivers wired to the 86Duino I/O pins	Hardware pulse generator (DDA in the interrupt service routine)
Custom Machine	CUSTOM_MACHINE_0, CUSTOM_MACHINE_1, CUSTOM_MACHINE_2 (100~102)	Remote actuators reached over a fieldbus (EtherCAT, Modbus, ...)	Target positions handed to user callbacks or a plugin every cycle through <code>updateAxes()</code>

Both kinds share the same motion API: `line()`, `arcXY()`, `gcode()`, JOG / MPG, feedrate override, software limits and electronic gearing work identically. Only the way positions reach the motors differs.

`EcatMachine` (Chapter 2.2) is a ready-made Custom Machine plugin for EtherCAT servo drives built on 86EVA. Chapter 2.3 describes how to build your own Custom Machine with callbacks or by deriving from `Machine`.

The life-cycle of every machine is the same:

```
construct -> configure (config_* / attach_* / set*) -> machineOn() ->
motion / JOG / MPG -> machineOff()
```

- All `config_*` and `attach_*` methods must be called **before** `machineOn()`; they return `false` once the machine is on.
- Motion methods (`line()`, `gcode()`, `home()`, ...) are accepted only **after** `machineOn()`.
- Only one `Machine` instance per axis group is allowed; check `isValid()` after construction.
- Positions and feedrates are expressed in user units (normally mm and mm/s). The pulses-per-unit scaling is set per axis with `config_PPU()`. Angles are in radians.

1.4 Hardware Wiring

1.4.1 Local Machine Pin Assignment

Each local axis group outputs a Pulse (Step) and a Direction signal per axis. The number of available axis groups depends on the 86Duino board:

Axis Group	Signal	Zero	One	EduCake
Machine 0	X-DIR	~42	~42	~42
	X-Pulse	~9	~9	~9
	Y-DIR	~43	~43	~43
	Y-Pulse	~6	~6	~6
	Z-DIR	~44	~44	~44
	Z-Pulse	~5	~5	~5
Machine 1	X-DIR	—	~18	~18
	X-Pulse	—	~13	~13
	Y-DIR	—	~19	~19
	Y-Pulse	—	~11	~11
	Z-DIR	—	~20	~20
	Z-Pulse	—	~10	~10
Machine 2	X-DIR	—	~33	—
	X-Pulse	—	~29	—
	Y-DIR	—	~34	—
	Y-Pulse	—	~30	—
	Z-DIR	—	~35	—
	Z-Pulse	—	~31	—

"—" means the axis group is not available on that board. Limit switches, jog buttons and hand-wheel encoders use ordinary digital / encoder pins chosen by the user.

1.4.2 Wiring Examples

The diagrams below show a typical connection of three A4988-type STEP/DIR stepper drivers to axis group 0 of each board. The library examples in this manual use pin 12 as a shared driver-enable output.

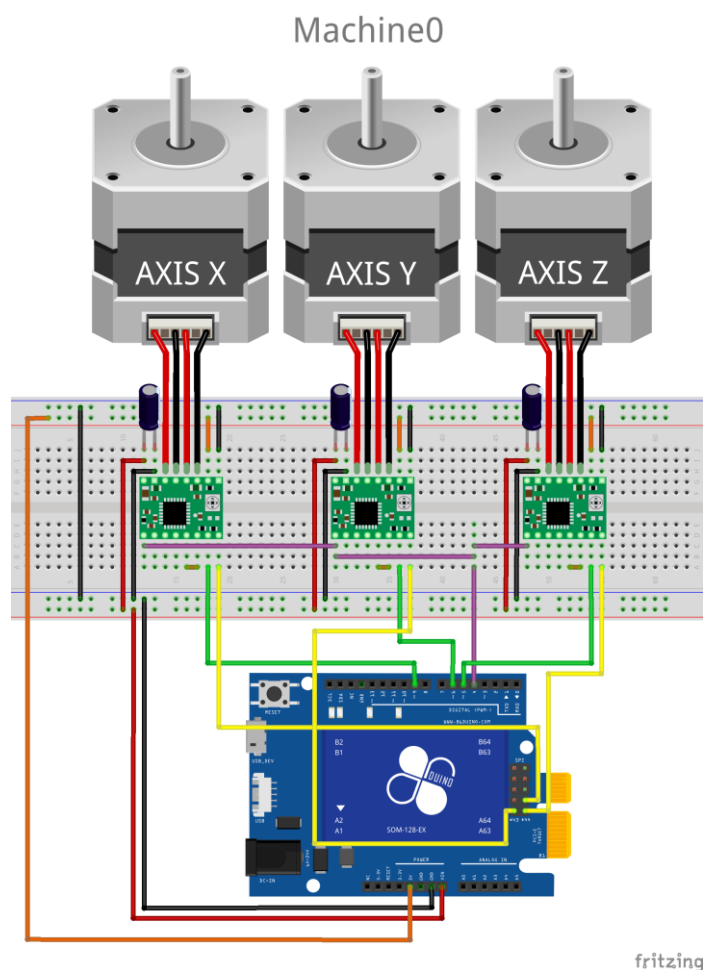


Figure 1-1 86Duino Zero – Machine 0

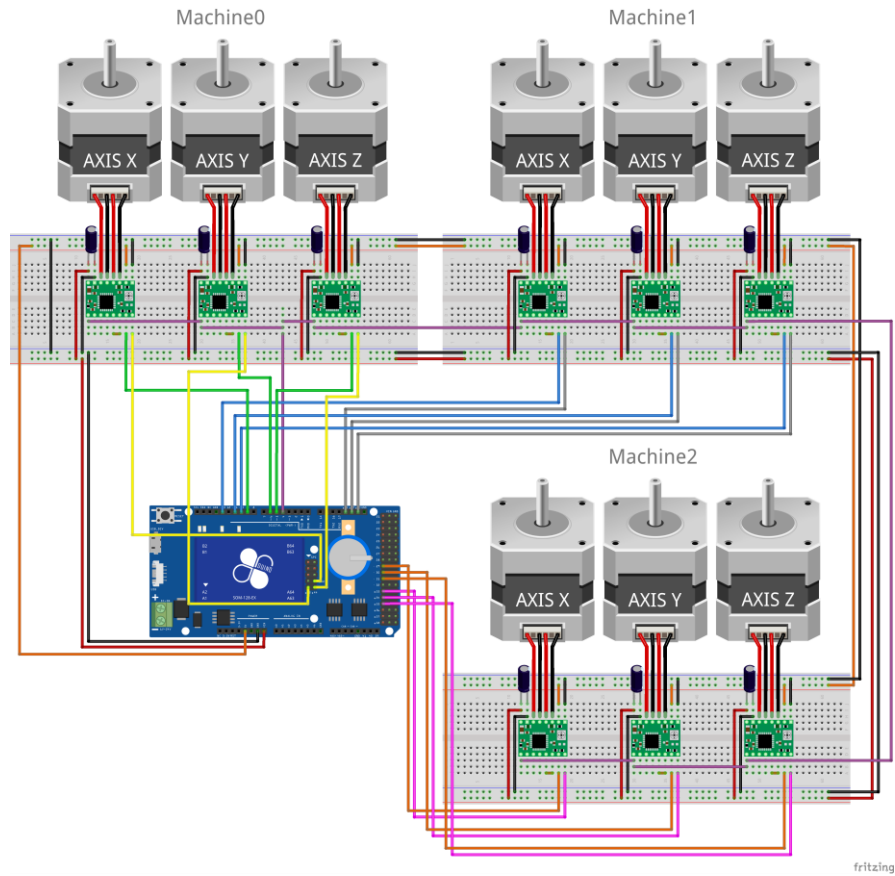


Figure 1-2 86Duino One – Machine 0

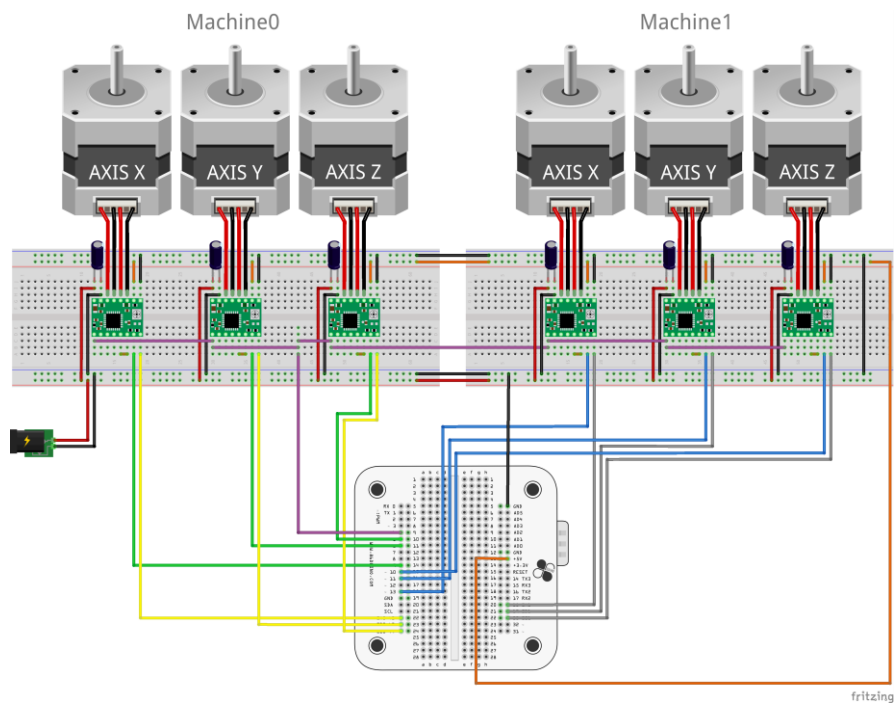


Figure 1-3 86Duino EduCake – Machine 0

1.4.3 EtherCAT Servo Configuration

For `EcatMachine`, the servo drives, I/O and hand wheel are EtherCAT slaves configured in **86EVA** (EtherCAT-based Virtual Arduino).

The 86EVA tool generates `myeva.h`, which declares the virtual devices (`VirtualServo1`, `VirtualMpg1`, digital inputs of the Virtual Arduino, ...) that `EcatMachine` binds to its axes. A typical set-up is a QEC-RXXMP3S or CiA 402 servo slave mapped to `VirtualServo1~VirtualServo3`.

Please refer to the 86EVA documentation on the QEC website for the slave configuration procedure.

1.5 Motion Methods

Motion86 provides four families of interpolated motion. Every motion method returns `true` when the block was queued and `false` when the command is not allowed. If the `feedrate` argument is omitted, the last recorded feedrate is used.

Motion	Native API	G-code
Linear	<code>line()</code>	G1
Arc	<code>arcXY()</code> , <code>arcXZ()</code> , <code>arcYZ()</code> (center / radius mode) <code>arcXY_Theta()</code> , <code>arcXZ_Theta()</code> , <code>arcYZ_Theta()</code> (angle mode)	G2 / G3 with I J K or R
Full circle	<code>circleXY()</code> , <code>circleXZ()</code> , <code>circleYZ()</code>	G2 / G3 with end point = start point, optional P
Helix	<code>helicalXY()</code> , <code>helicalXZ()</code> , <code>helicalYZ()</code>	G2 / G3 with the third axis and P

Direction convention. Clockwise (CW) and counter-clockwise (CCW) are judged looking from the positive direction of the axis perpendicular to the plane: Z for XY, Y for XZ, X for YZ. `revDir = false` selects CW and `revDir = true` selects CCW. In the `_Theta` and `helical` methods a positive angle is CW and a negative angle is CCW.

1.5.1 Linear Motion

The machine moves in a straight line to the target position (Figure 1-4).

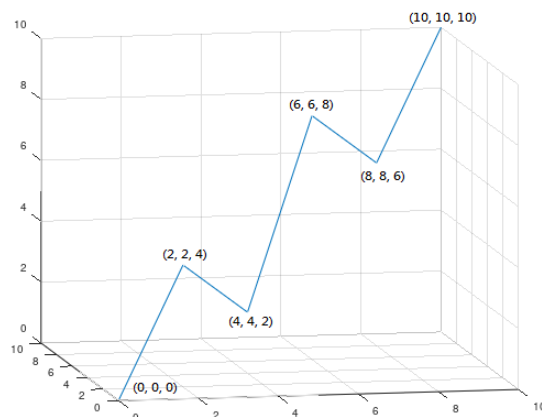


Figure 1-4 Linear motion

```
machine.gcode("G1 X0 Y0 Z0 F600");
machine.gcode("G1 X2 Y2 Z4 F600");
machine.gcode("G1 X4 Y4 Z2 F600");
machine.gcode("G1 X6 Y6 Z8 F600");
machine.gcode("G1 X8 Y8 Z6 F600");
machine.gcode("G1 X10 Y10 Z10 F600");
```

// or, with the native API:

```
machine.line(0, 0, 0, 600);
machine.line(2, 2, 4, 600);
machine.line(4, 4, 2, 600);
machine.line(6, 6, 8, 600);
machine.line(8, 8, 6, 600);
machine.line(10, 10, 10, 600);
```

1.5.2 Arc Motion

An arc can be described in three ways:

- **Center mode** – the center point and the end point are given.
- **Radius mode** – the radius and the end point are given.
- **Angle (theta) mode** – the center point and the angle to sweep are given; the end point is computed.

Whichever mode is used, the direction must be chosen: `G2 / revDir = false` for CW (Figure 1-5) and `G3 / revDir = true` for CCW (Figure 1-6).

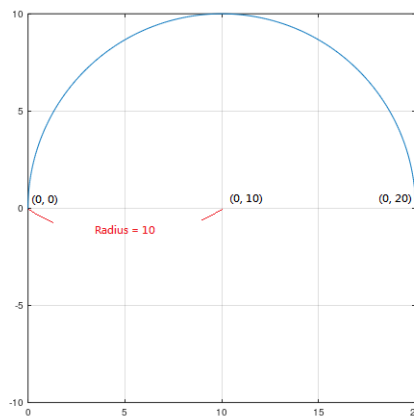


Figure 1-5 Clockwise arc (G2)

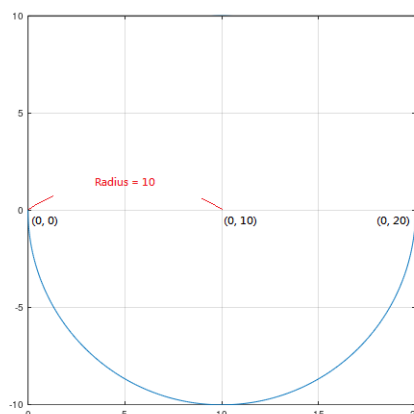


Figure 1-6 Counter-clockwise arc (G3)

```
// Center mode: from (0,0) to (0,20) around center (0,10)
machine.gcode("G2 X0 Y20 I0 J10 F600");    // CW (Figure 1-5)
machine.gcode("G3 X0 Y20 I0 J10 F600");    // CCW (Figure 1-6)
machine.arcXY(0, 10, 0, 20, false, 600);   // CW
machine.arcXY(0, 10, 0, 20, true, 600);    // CCW
```

Radius mode. For a given radius and end point (other than a 180° arc) four paths exist (Figure 1-7). The direction is chosen with G2 / G3 or revDir; the **sign of the radius** selects the short arc (< 180°, positive R) or the long arc (> 180°, negative R).

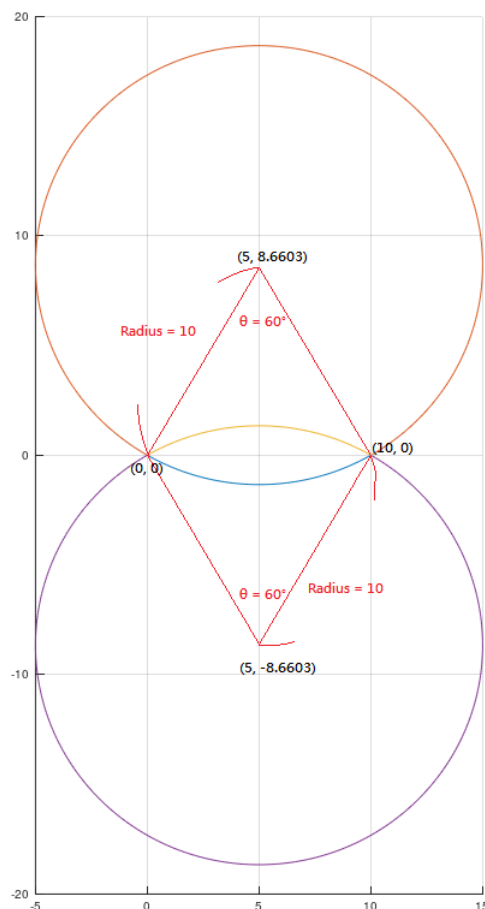


Figure 1-7 The four arcs of radius 10 to the same end point

Path	Direction	Sweep	G-code	Native API
Yellow	CW	< 180°	G2 X10 Y0 R10 F600	arcXY(10, 10, 0, false, 600)
Orange	CW	> 180°	G2 X10 Y0 R-10 F600	arcXY(-10, 10, 0, false, 600)
Blue	CCW	< 180°	G3 X10 Y0 R10 F600	arcXY(10, 10, 0, true, 600)
Purple	CCW	> 180°	G3 X10 Y0 R-10 F600	arcXY(-10, 10, 0, true, 600)

Angle mode. `arcXY_Theta(cX, cY, theta, feedrate)` sweeps `theta` radians around the center; the sign of the angle selects the direction.

```
// 90-degree CW arc around (10, 0) at F600  
machine.arcXY_Theta(10, 0, HALF_PI, 600);  
// 90-degree CCW arc around (10, 0) at F600  
machine.arcXY_Theta(10, 0, -HALF_PI, 600);
```

Note: In center and radius mode the end point must lie on the circle defined by the parameters. If it does not, the resulting path is undefined and the machine may move unexpectedly. Check the geometry before sending the command, especially when generating G-code programmatically.

1.5.3 Circle Motion

When the end point of an arc equals the current position, Motion86 draws a full circle. In G-code the P word repeats the circle the given number of times. The native `circleXY()` / `circleXZ()` / `circleYZ()` methods draw one full circle and return to the starting point.

```
// Two full CW turns around (10, 0) at F600, ending back at (0, 0)
machine.gcode("G2 X0 Y0 I10 J0 P2 F600");

// One full CW circle around (10, 0), back to the current position
machine.circleXY(10, 0, false, 600);
```

***Note:** In center or radius mode, when the start and end position coincide and the geometry is otherwise ambiguous (infinite solutions), no motion is performed.

1.5.4 Helical (Spiral) Motion

A helix combines an arc in one plane with a linear move of the third axis (Figures 1-8 to 1-10). The plane is selected by the method name (or by G17 / G18 / G19 in G-code) and the sign of the angle selects the direction.

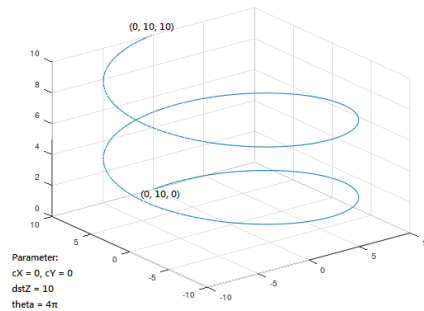


Figure 1-8 helicalXY

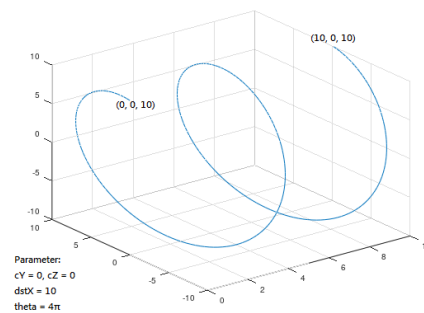


Figure 1-9 helicalYZ

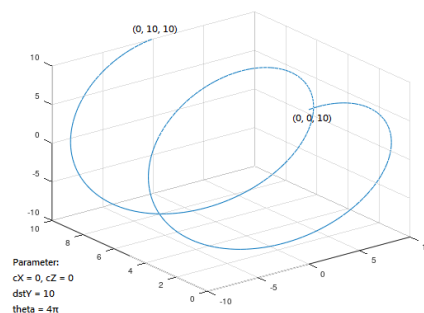


Figure 1-10 helicalXZ

```
// Two turns (4*PI rad) around (0, 0) in XY while Z travels to 10, F600
machine.helicalXY(0, 0, 10, 4 * PI, 600);
// Two turns around (0, 0) in YZ while X travels to 10
machine.helicalYZ(0, 0, 10, 4 * PI, 600);
// Two turns around (0, 0) in XZ while Y travels to 10
machine.helicalXZ(0, 0, 10, 4 * PI, 600);
```

1.6 Architecture

The `Machine` object is the core controller. Every motion command passes through four layers:

- **G-code layer** – parses G-code strings (`gcode()`) or receives native API calls and converts them into motion blocks.
- **Planner** – queues the blocks (1024 entries), performs speed look-ahead and computes the acceleration / deceleration profile.
- **Interpolator** – runs every 0.5 ms and produces per-axis position increments.
- **Output stage** – on a Local Machine the **DDA** interrupt service routine emits the pulses to the I/O pins in real time; on a Custom Machine `updateAxes()` publishes the new target of each axis through `setAxisPosition()` (plugin) or the `setCB` callback, and reads the actual position back through `getAxisPosition()` / `getCB`.

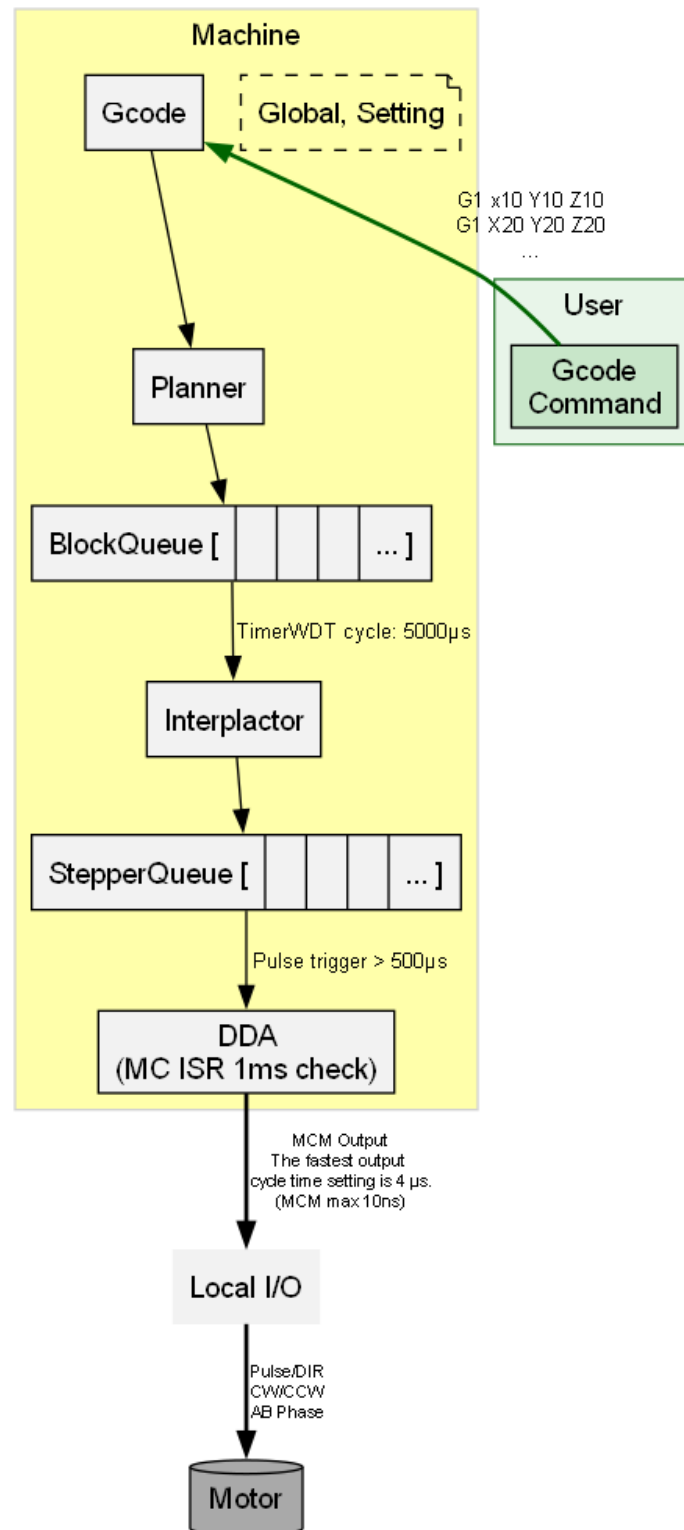


Figure 1-11 Local Machine data flow

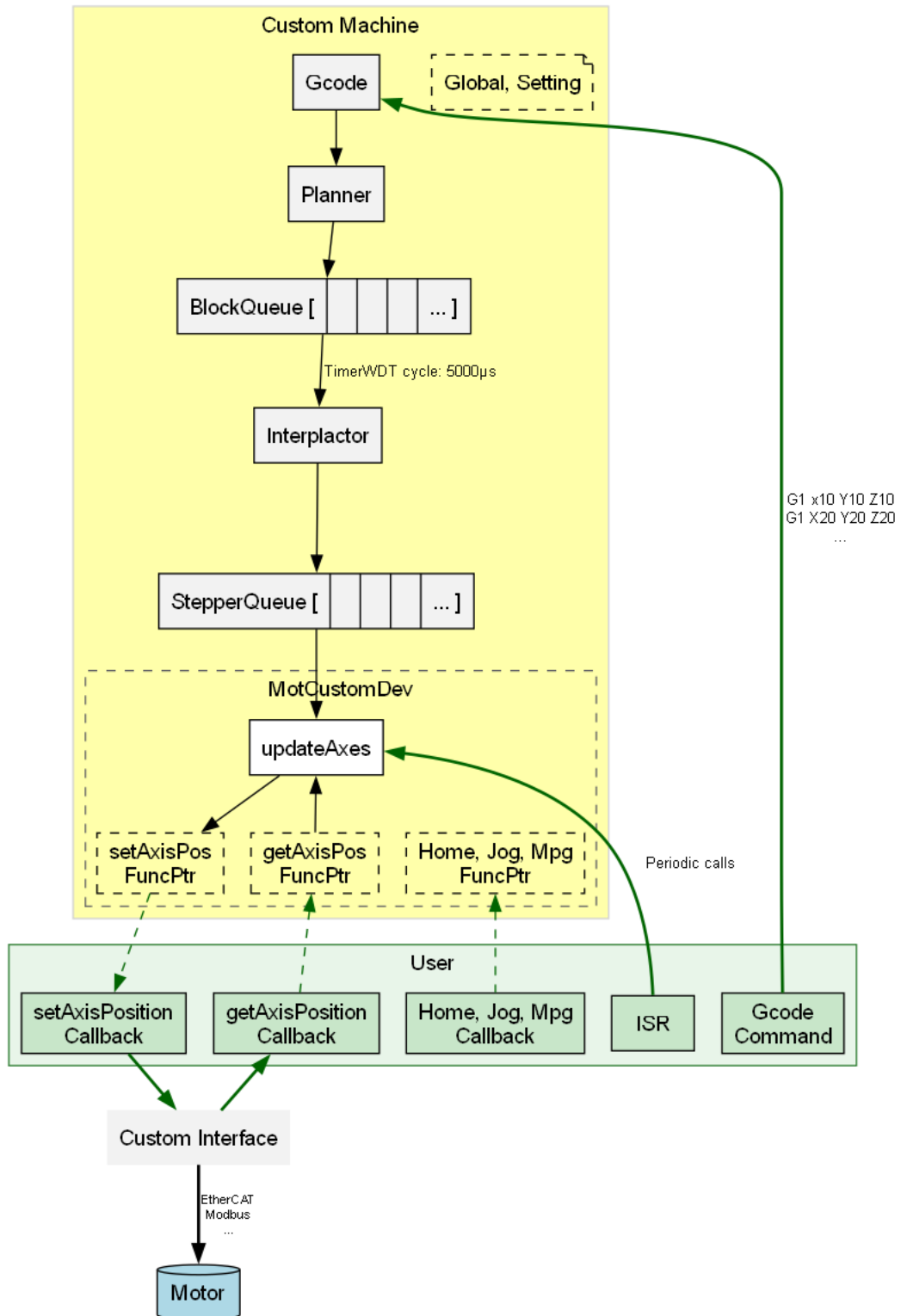


Figure 1-12 Custom Machine data flow (callbacks)

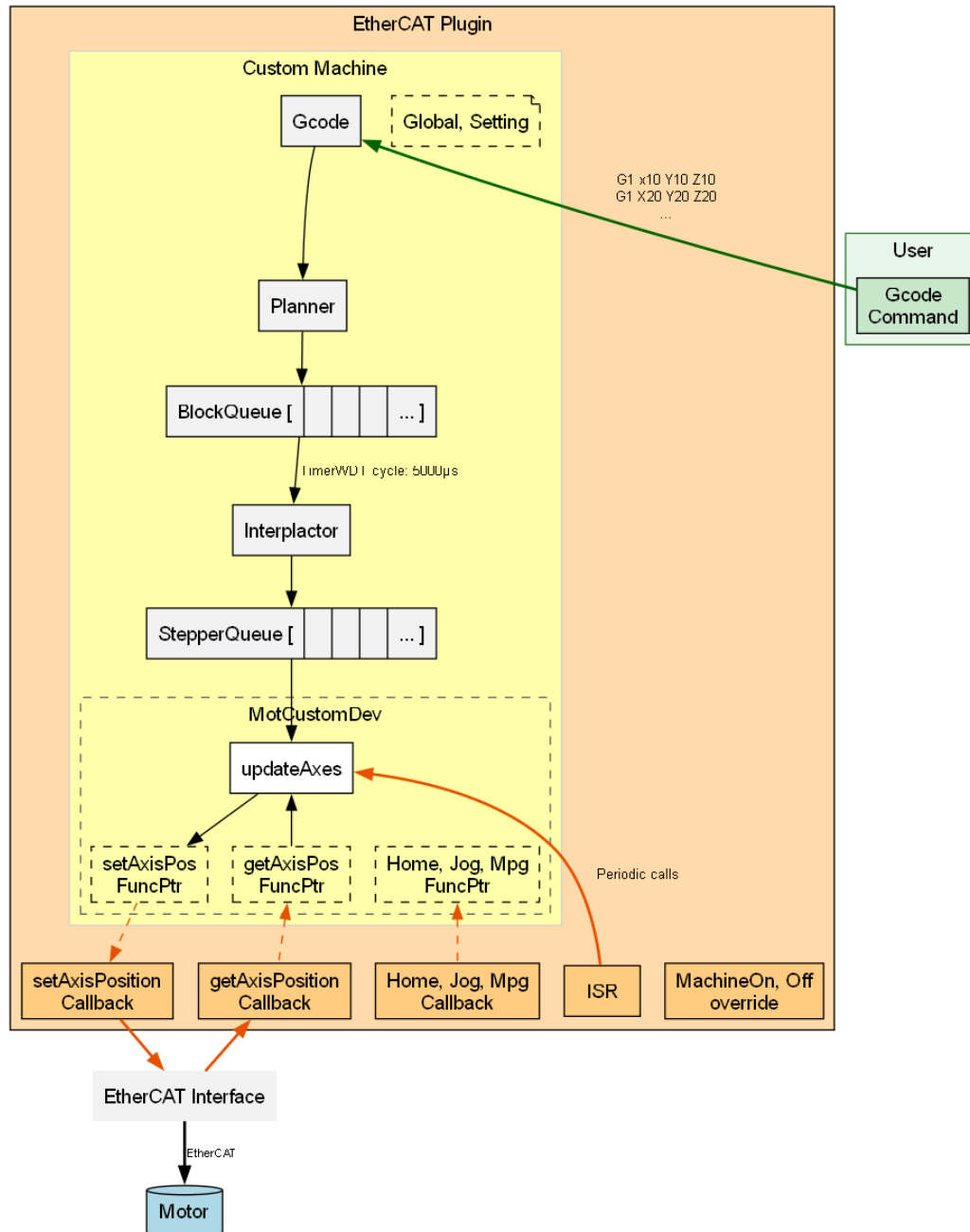


Figure 1-13 EcatMachine plugin data flow

1.7 Functions Table

This section briefly lists the Motion86 API.
Detailed descriptions are given in Chapter 2.

1.7.1 Machine Class

Machine is the core motion controller class. It is used directly for local machines and is the base class of every Custom Machine plugin.

Basic Method

Function Name	Description
Machine()	Constructs a Machine object bound to one axis group.
~Machine()	Destroys the Machine object and releases its resources, including the axis group, so that a new Machine can be created on the same group later.
machineOn()	Switches the Machine object to *MachineOn* mode.
machineOff()	Switches the Machine object to *MachineOff* mode.
home()	Returns the machine to its home point.
gcode()	Sends one G-code command string to the machine.
stop()	Stops the machine immediately and clears all queued motion blocks and planning data.
emgStop()	Puts the machine into the emergency-stop state: all motion control and planning data are cleared, and the machine refuses every command until the state is cleared with clearEMGStop().
clearEMGStop()	Clears the emergency-stop state so that the machine accepts commands again.

Basic Control

Function Name	Description
line()	Moves in a straight line to the target point at the given feedrate.
arcXY()	Moves along a clockwise or counter-clockwise arc in the XY plane.
arcXZ()	Moves along a clockwise or counter-clockwise arc in the XZ plane.
arcYZ()	Moves along a clockwise or counter-clockwise arc in the YZ plane.
arcXY_Theta()	Moves along an arc in the XY plane using angle mode : the center point and the angle to sweep are given, and the end point is computed.
arcXZ_Theta()	Moves along an arc in the XZ plane using angle mode : the center point and the angle to sweep are given, and the end point is computed.
arcYZ_Theta()	Moves along an arc in the YZ plane using angle mode : the center point and the angle to sweep are given, and the end point is computed.

Function Name	Description
circleXY()	Draws one complete circle in the XY plane around the given center and returns to the starting point.
circleXZ()	Draws one complete circle in the XZ plane around the given center and returns to the starting point.
circleYZ()	Draws one complete circle in the YZ plane around the given center and returns to the starting point.
helicalXY()	Moves along a helix: a circular motion of theta radians in the XY plane around the given center, combined with a linear move of the Z axis to dstZ.
helicalXZ()	Moves along a helix: a circular motion of theta radians in the XZ plane around the given center, combined with a linear move of the Y axis to dstY.
helicalYZ()	Moves along a helix: a circular motion of theta radians in the YZ plane around the given center, combined with a linear move of the X axis to dstX.

Write Configuration

Function Name	Description
config_MaxVelocity()	Sets the maximum speed of the specified axis.
config_MaxAcc()	Sets the maximum acceleration used by the planner for the machine.
config_PosLimit()	Sets the software travel limits of the specified axis.
config_PPU()	Sets the Pulses per Unit (PPU) of the specified axis.
config_ReverseDirection()	Reverses the direction of motion of the specified axis.
config_MaxPulseSpeed()	Sets the maximum pulse output frequency of the machine.
config_PulseMode()	Sets the pulse output mode (servo / stepper driver signal type) of the specified axis.
config_HomePins()	Sets the pins of the hardware limit switches used to define the home point of the machine.
setDefaultFeedrate()	Sets the default feedrate of the machine.
setHomeSpeed()	Sets the feedrate used by home().
setRelative()	Switches the machine to relative (incremental) positioning: the target coordinates of the motion methods are interpreted as increments from the current position.
setAbsolute()	Switches the machine to absolute positioning (default): the target coordinates of the motion methods are absolute positions relative to the home point.
enableSoftLimit()	Enables the software travel limits set with config_PosLimit().
disableSoftLimit()	Disables the software travel limits.

Read Configuration

Function Name	Description
getMaxVelocity()	Returns the maximum speed limit of the specified axis.

Function Name	Description
getMaxAcc()	Returns the maximum acceleration limit of the machine.
getPPU()	Returns the PPU (pulses per unit) of the specified axis.
getActualPos()	Returns the current position of the machine in the workspace, either for one axis or for all three axes at once through reference arguments.
getPulseMode()	Returns the pulse output mode of the specified axis.
getCmdCount()	Returns the number of motion blocks currently queued in the command buffer.
getFreeCmdCount()	Returns the number of free entries in the command buffer.
isDirectionReversed()	Reports whether the specified axis has been configured to move in the reverse direction.
isMoving()	Reports whether the machine is currently moving or still has motion blocks to execute.
isValid()	Reports whether the Machine object exists and was created successfully.
isCmdBufferFull()	Reports whether the command buffer (1024 blocks) is full.

JOG / MPG Control

Function Name	Description
beginJog()	Enters JOG mode.
endJog()	Leaves JOG mode and returns to normal motion control.
setJogSpeed()	Sets the feedrate used in JOG / incJog mode.
setJogOffset()	Sets the distance moved per switch press in incJog mode.
setJogPins()	Assigns the jog switch pins of each axis direction without entering JOG mode.
beginMpg()	Enters MPG (hand-wheel) mode.
endMpg()	Leaves MPG mode and returns to normal motion control.
setMpgOn()	Re-enables hand-wheel following while the machine stays in MPG mode.
setMpgOff()	Temporarily disables hand-wheel following while the machine stays in MPG mode.
setMpgSpeed()	Sets the feedrate used in MPG mode.
setMpgAxis()	Selects the axis driven by the hand wheel in MPG mode.
setMpgRatio()	Sets the multiplier between the encoder count change and the travelled distance in MPG mode.
getJogPos()	Returns the target position of the machine in JOG / MPG mode, either for one axis or for all three axes through reference arguments.

Feedrate Override

Function Name	Description
feedrateOverride()	Rescales the feedrate of all motion blocks in the buffer that have not been executed yet, and recalculates the trajectory.
getFeedrateOverride()	Returns the current feedrate override ratio.

Axis Following (Electronic Gear)

Function Name	Description
<code>config_EGearSlave()</code>	Registers another Machine as the electronic-gear slave of this machine.

Calculation Tools

Function Name	Description
<code>computePPU_Belt()</code>	Computes the PPU (pulses per mm) of a belt-driven axis from the motor pulses per revolution, the belt pitch and the pulley tooth count.
<code>computePPU_LeadScrew()</code>	Computes the PPU (pulses per mm) of a lead-screw axis from the motor pulses per revolution, the screw pitch and the gearbox ratio.

1.7.2 EcatMachine Class

`EcatMachine` inherits from `Machine` and drives EtherCAT servo drives through 86EVA.

In addition to all `Machine` methods it provides:

Function Name	Description
<code>EcatMachine()</code>	Constructs an EtherCAT motion controller on a custom axis group.
<code>~EcatMachine()</code>	Destroys the <code>EcatMachine</code> object.
<code>attach_EcatServo_X()</code>	Binds a servo drive to the X axis.
<code>attach_EcatServo_Y()</code>	Binds a servo drive to the Y axis.
<code>attach_EcatServo_Z()</code>	Binds a servo drive to the Z axis.
<code>attach_EcatHomePin_X()</code>	Assigns a Virtual Arduino digital input as the home switch of the X axis.
<code>attach_EcatHomePin_Y()</code>	Assigns a Virtual Arduino digital input as the home switch of the Y axis.
<code>attach_EcatHomePin_Z()</code>	Assigns a Virtual Arduino digital input as the home switch of the Z axis.
<code>attach_EcatJogPins_X()</code>	Assigns two Virtual Arduino digital inputs as the positive and negative jog switches of the X axis, each with its own active level.
<code>attach_EcatJogPins_Y()</code>	Assigns two Virtual Arduino digital inputs as the positive and negative jog switches of the Y axis, each with its own active level.
<code>attach_EcatJogPins_Z()</code>	Assigns two Virtual Arduino digital inputs as the positive and negative jog switches of the Z axis, each with its own active level.
<code>attach_EcatMpg()</code>	Attaches the hand-wheel source used in MPG mode: an 86EVA virtual MPG (<code>MyVirtualMpg</code>) or a virtual encoder (<code>MyVirtualEncoder</code>), by reference or by pointer.
<code>machineOn()</code> [<code>EcatMachine</code>]	Overrides <code>Machine::machineOn()</code> and performs the complete EtherCAT start-up:
<code>machineOff()</code> [<code>EcatMachine</code>]	Overrides <code>Machine::machineOff()</code> : leaves <code>MachineOn</code> mode and then disables every attached servo.

1.7.3 Custom Machine Tools

Methods of `Machine` used to connect a Custom Machine to its actuators and I/O.

Function Name	Description
<code>config_CustomActuators_Callbacks()</code>	Registers the position write / read callbacks of a Custom Machine.
<code>config_CustomActuators_CycleTime()</code>	Declares the period at which <code>updateAxes()</code> will be called for a Custom Machine.
<code>config_HomeController_Callback()</code>	Registers the callback that <code>home()</code> uses to read the home switches of a Custom Machine.
<code>config_JogController_Callback()</code>	Registers the callback that JOG mode uses to read the jog switches of a Custom Machine.
<code>config_MpgController_Callback()</code>	Registers the callback that MPG mode uses to read the hand-wheel counter of a Custom Machine.
<code>updateAxes()</code>	Advances a Custom Machine by one interpolation cycle: pushes the new target position of every axis (through <code>setAxisPosition()</code> or the <code>setCB</code> callback) and reads back the actual positions (through <code>getAxisPosition()</code> or <code>getCB</code>).



Ch. 2

Functions

86Duino Coding IDE 501+.

Motion86 Library

2.1 Machine Class

The `Machine` class provides the configuration and control methods of an XYZ axis group.
`#include "Motion86.h"` to use it.

Groups:

- [2.1.1 Basic Method](#)
- [2.1.2 Basic Control](#)
- [2.1.3 Write Configuration](#)
- [2.1.4 Read Configuration](#)
- [2.1.5 JOG / MPG Control](#)
- [2.1.6 Feedrate Override](#)
- [2.1.7 Axis Following \(Electronic Gear\)](#)
- [2.1.8 Calculation Tools](#)

2.1.1 Basic Method

Object life-cycle, G-code entry point, homing and stop functions.

Functions:

- [Machine\(\)](#)
- [~Machine\(\)](#)
- [machineOn\(\)](#)
- [machineOff\(\)](#)
- [home\(\)](#)
- [gcode\(\)](#)
- [stop\(\)](#)
- [emgStop\(\)](#)
- [clearEMGStop\(\)](#)

Machine()

Description

Constructs a `Machine` object bound to one axis group. Only one `Machine` instance per axis group is allowed.

Local axis groups (`MACHINE_0~MACHINE_2`) drive stepper / pulse-input servo drivers through the 86Duino local I/O. Custom axis groups (`CUSTOM_MACHINE_0~CUSTOM_MACHINE_2`) drive remote actuators (EtherCAT, Modbus, ...) and require a Custom Machine implementation – either the callback functions described in 2.3, or a plugin class such as `EcatMachine`.

An integer overload is provided for convenience; the value is cast to `axisGroup_t`.

Syntax

```
Machine machine(axis_group);
```

```
Machine machine((int)axis_group);
```

Parameters

- `axis_group`: Axis group to control. Default is `MACHINE_0`.
- `MACHINE_0` (0), `MACHINE_1` (1), `MACHINE_2` (2): local machines.
- `CUSTOM_MACHINE_0` (100), `CUSTOM_MACHINE_1` (101), `CUSTOM_MACHINE_2` (102): custom machines.

Returns

None. Use `isValid()` to check whether the object was created successfully.

Example

```
#include "Motion86.h"

// Local machine 0 (pulse output on the 86Duino pins of axis group 0).
Machine machine(0);

// Local machine 1, written with the enumeration.
Machine machine1(MACHINE_1);

void setup() {
  while (!Serial);
  if (!machine.isValid()) {
    Serial.println("Machine 0 was not created.");
    while (1);
  }
  // Configure and turn on the machines here...
}

void loop() {
}
```

~Machine()

Description

Destroys the `Machine` object and releases its resources, including the axis group, so that a new `Machine` can be created on the same group later.

The destructor is virtual and can be overridden by a custom plugin.

Syntax

```
delete machine;
```

Parameters

- `machine`: a pointer to a `Machine` object created with `new`.

Returns

None.

Example

```
#include "Motion86.h"

Machine *machine = NULL;

void setup() {
  while (!Serial);
  // Construct the machine dynamically.
  machine = new Machine(0);
  machine->config_PPU(AXIS_X, 80.0);
  machine->config_PPU(AXIS_Y, 80.0);
  machine->config_PPU(AXIS_Z, 1600.0);
  machine->machineOn();

  machine->line(10, 10, 0, 400);
  while (machine->isMoving());

  // Release the axis group. It can be re-used by a new Machine object.
  machine->machineOff();
  delete machine;
  machine = NULL;
}

void loop() {
}
```

machineOn()

Description

Switches the `Machine` object to `*MachineOn*` mode.

The machine must be in MachineOn mode before any motion control method can be used.

Configuration methods (`config_*`) are no longer accepted once the machine is on.

This method is virtual and can be overridden by a custom plugin to customise the start-up sequence; `EcatMachine::machineOn()` uses this to initialise the EtherCAT drives.

Syntax

```
machine.machineOn();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 3200.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
  machine.setDefaultFeedrate(400);

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
```

```
machine.line(10, 10, 0);

// The machine must be turned off before it can be re-configured.
machine.machineOff();

// Reverse the direction of an axis if required.
// In this example the X and Y axes need to be reversed.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// The machine must be turned on before it can be controlled.
machine.machineOn();

machine.line(20, 20, 0);
}

void loop() {

}
```

machineOff()

Description

Switches the `Machine` object to `*MachineOff*` mode. Motion control is stopped and the machine can be re-configured.

This method is virtual and can be overridden by a custom plugin to customise the shutdown sequence.

Syntax

```
machine.machineOff();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 3200.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
  machine.setDefaultFeedrate(400);

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);

  machine.line(10, 10, 0);
```

```
// The machine must be turned off before it can be re-configured.  
machine.machineOff();  
  
// Reverse the direction of an axis if required.  
// In this example the X and Y axes need to be reversed.  
machine.config_ReverseDirection(AXIS_X);  
machine.config_ReverseDirection(AXIS_Y);  
  
// The machine must be turned on before it can be controlled.  
machine.machineOn();  
  
machine.line(20, 20, 0);  
}  
  
void loop() {  
  
}
```

home()

Description

Returns the machine to its home point.

If `config_HomePins()` has been configured, each axis moves in the negative direction until its limit switch is triggered, and that position becomes the home point (position 0).

If `config_HomePins()` has not been configured, the current position is taken as the home point.

On a Custom Machine, the home switches are read through the home callback

(`config_HomeController_Callback()`) or the plugin's `homeIoHelper()`; the homing hardware must be declared first (for `EcatMachine`, at least one `attach_EcatHomePin_*`()), otherwise the homing procedure is skipped.

Syntax

```
machine.home();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

  while (!Serial);
  pinMode(EnablePin, OUTPUT);
  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);
  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
  // Set the software limits of the machine.
  machine.config_PosLimit(AXIS_X, 0, 300);
  machine.config_PosLimit(AXIS_Y, 0, 200);
  machine.config_PosLimit(AXIS_Z, 0, 300);
}
```

```
// Set the limit-switch pins that define the home point.
machine.config_HomePins(2, 7, 8);
// The machine must be turned on before it can be controlled.
machine.machineOn();
// Set the default feedrate.
machine.setDefaultFeedrate(400);
// Enable the software limits.
machine.enableSoftLimit();
// Set the homing feedrate.
machine.setHomeSpeed(1000, 1000, 200);
// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
// Return to the home point defined by the limit switches.
machine.home();

}

void loop() {
  // Motion control code...
}
```

gcode()

Description

Sends one G-code command string to the machine. The supported commands (G1, G2, G3, G4, G17~G19, G90/G91, G90.1/G91.1, G92) are listed in Appendix A.1 G-code Command Reference. Feed the command buffer with `isCmdBufferFull()` / `getFreeCmdCount()` when streaming a G-code file; the buffer holds 1024 blocks.

Syntax

```
machine.gcode(cmd);
```

Parameters

- `machine`: a `Machine` object.
- `cmd`: the G-code command string, e.g. "G1 X10 Y10 F500".

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
```

```
// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Move left to (-10, 0).
  machine.gcode("G1 X-10 Y0 F400");

  // Clockwise 90-degree arc with radius 10 to (0, 10).
  machine.gcode("G2 X0 Y10 R10 F400");

  // Move down to (0, -10).
  machine.gcode("G1 X0 Y-10 F400");

  // Counter-clockwise 90-degree arc with radius 10 to (10, 0).
  machine.gcode("G3 X10 Y0 R10 F400");

  // Move back to (0, 0).
  machine.gcode("G1 X0 Y0 F400");

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

stop()

Description

Stops the machine immediately and clears all queued motion blocks and planning data. The machine stays in MachineOn mode and accepts new commands right away.

Syntax

```
machine.stop();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
  machine.setDefaultFeedrate(400);

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
```

```
}

void loop() {
  // Move left to (-10, 0).
  machine.line(-10, 0, 0);

  // Radius 10, counter-clockwise to (X=0, Y=10).
  machine.arcXY(10, 0, 10, true);

  // Move down to (0, -10).
  machine.line(0, -10, 0);

  // Radius 10, clockwise to (X=10, Y=0).
  machine.arcXY(10, 10, 0, false);

  // Move back to (0, 0).
  machine.line(0, 0, 0);

  // The motion will be stopped after 1 second.
  delay(1000);
  machine.stop();
}
```

emgStop()

Description

Puts the machine into the emergency-stop state: all motion control and planning data are cleared, and the machine refuses every command until the state is cleared with `clearEMGStop()`.

Syntax

```
machine.emgStop();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
  machine.setDefaultFeedrate(400);

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
```

```
}

void loop() {
  // Move left to (-10, 0).
  machine.line(-10, 0, 0);

  // Radius 10, counter-clockwise to (X=0, Y=10).
  machine.arcXY(10, 0, 10, true);

  // Move down to (0, -10).
  machine.line(0, -10, 0);

  // Radius 10, clockwise to (X=10, Y=0).
  machine.arcXY(10, 10, 0, false);

  // Move back to (0, 0).
  machine.line(0, 0, 0);

  // The motion will be stopped after 1 second.
  delay(1000);
  machine.emgStop();
  machine.clearEMGStop();
}
```

clearEMGStop()

Description

Clears the emergency-stop state so that the machine accepts commands again.

Syntax

```
machine.clearEMGStop();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
  machine.setDefaultFeedrate(400);

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}
```

```
void loop() {  
  // Move left to (-10, 0).  
  machine.line(-10, 0, 0);  
  
  // Radius 10, counter-clockwise to (X=0, Y=10).  
  machine.arcXY(10, 0, 10, true);  
  
  // Move down to (0, -10).  
  machine.line(0, -10, 0);  
  
  // Radius 10, clockwise to (X=10, Y=0).  
  machine.arcXY(10, 10, 0, false);  
  
  // Move back to (0, 0).  
  machine.line(0, 0, 0);  
  
  // The motion will be stopped after 1 second.  
  delay(1000);  
  machine.emgStop();  
  machine.clearEMGStop();  
}
```

2.1.2 Basic Control

Interpolated motion: linear, arc, circle and helical moves.

Functions:

- [line\(\)](#)
- [arcXY\(\)](#)
- [arcXZ\(\)](#)
- [arcYZ\(\)](#)
- [arcXY_Theta\(\)](#)
- [arcXZ_Theta\(\)](#)
- [arcYZ_Theta\(\)](#)
- [circleXY\(\)](#)
- [circleXZ\(\)](#)
- [circleYZ\(\)](#)
- [helicalXY\(\)](#)
- [helicalXZ\(\)](#)
- [helicalYZ\(\)](#)

line()

Description

Moves in a straight line to the target point at the given feedrate. See 1.5 Motion Methods for details.

Syntax

```
machine.line(x, y, z);
machine.line(x, y, z, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `x`: target X coordinate.
- `y`: target Y coordinate.
- `z`: target Z coordinate.
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
```

```
// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Linear move to (10, 10, 10).
  machine.line(10, 10, 10);

  // Linear move to (0, 0, 0).
  machine.line(0, 0, 0);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

arcXY()

Description

Moves along a clockwise or counter-clockwise arc in the XY plane. The direction is judged looking from the positive direction of the Z axis.

Two arc modes are selected by the arguments:

1. **Center mode** — the center point and the end point are given.
2. **Radius mode** — the radius and the end point are given. A positive radius selects the arc shorter than 180°, a negative radius the arc longer than 180°.

See 1.5 Motion Methods for the geometry of each mode.

Syntax

```
machine.arcXY(r, dstX, dstY);
machine.arcXY(r, dstX, dstY, revDir);
machine.arcXY(r, dstX, dstY, revDir, feedrate);
machine.arcXY(cX, cY, dstX, dstY);
machine.arcXY(cX, cY, dstX, dstY, revDir);
machine.arcXY(cX, cY, dstX, dstY, revDir, feedrate);
```

Parameters

- **machine**: a **Machine** object.
- **r**: radius of the arc in radius mode (negative = arc greater than 180°).
- **cX**: center X coordinate in center mode.
- **cY**: center Y coordinate in center mode.
- **dstX**: target X coordinate.
- **dstY**: target Y coordinate.
- **revDir**: Direction of rotation: **false** = clockwise (default), **true** = counter-clockwise.
- **feedrate**: Feedrate. If omitted, the last recorded feedrate is used.

Returns

bool: **true** if the motion block was queued; **false** if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Whichever arc mode is used, a wrong parameter set may leave the target point off the circle, which results in unexpected machine motion.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
```

```

while (!Serial);
pinMode(EnablePin, OUTPUT);

// Reverse the direction of an axis if required.
// In this example the X and Y axes need to be reversed.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// PPU (pulses per unit) is a virtual length unit defined by your application.
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Radius 10, counter-clockwise to (X=10, Y=10).
  machine.arcXY(10, 10, 10, true);

  // Radius 10, clockwise to (X=0, Y=0).
  machine.arcXY(10, 0, 0, false);

  // Center (X=0, Y=10), counter-clockwise to (X=10, Y=10).
  machine.arcXY(0, 10, 10, 10, true);

  // Center (X=0, Y=10), clockwise to (X=0, Y=0).
  machine.arcXY(0, 10, 0, 0, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}

```

arcXZ()

Description

Moves along a clockwise or counter-clockwise arc in the XZ plane. The direction is judged looking from the positive direction of the Y axis.

Two arc modes are selected by the arguments:

1. **Center mode** — the center point and the end point are given.
2. **Radius mode** — the radius and the end point are given. A positive radius selects the arc shorter than 180°, a negative radius the arc longer than 180°.

See 1.5 Motion Methods for the geometry of each mode.

Syntax

```
machine.arcXZ(r, dstX, dstZ);
machine.arcXZ(r, dstX, dstZ, revDir);
machine.arcXZ(r, dstX, dstZ, revDir, feedrate);
machine.arcXZ(cX, cZ, dstX, dstZ);
machine.arcXZ(cX, cZ, dstX, dstZ, revDir);
machine.arcXZ(cX, cZ, dstX, dstZ, revDir, feedrate);
```

Parameters

- **machine**: a **Machine** object.
- **r**: radius of the arc in radius mode (negative = arc greater than 180°).
- **cX**: center X coordinate in center mode.
- **cZ**: center Z coordinate in center mode.
- **dstX**: target X coordinate.
- **dstZ**: target Z coordinate.
- **revDir**: Direction of rotation: **false** = clockwise (default), **true** = counter-clockwise.
- **feedrate**: Feedrate. If omitted, the last recorded feedrate is used.

Returns

bool: **true** if the motion block was queued; **false** if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Whichever arc mode is used, a wrong parameter set may leave the target point off the circle, which results in unexpected machine motion.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
```

```

while (!Serial);
pinMode(EnablePin, OUTPUT);

// Reverse the direction of an axis if required.
// In this example the X and Y axes need to be reversed.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// PPU (pulses per unit) is a virtual length unit defined by your application.
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Radius 10, counter-clockwise to (X=10, Z=10).
  machine.arcXZ(10, 10, 10, true);

  // Radius 10, clockwise to (X=0, Z=0).
  machine.arcXZ(10, 0, 0, false);

  // Center (X=0, Z=10), counter-clockwise to (X=10, Z=10).
  machine.arcXZ(0, 10, 10, 10, true);

  // Center (X=0, Z=10), clockwise to (X=0, Z=0).
  machine.arcXZ(0, 10, 0, 0, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}

```

arcYZ()

Description

Moves along a clockwise or counter-clockwise arc in the YZ plane. The direction is judged looking from the positive direction of the X axis.

Two arc modes are selected by the arguments:

1. **Center mode** — the center point and the end point are given.
2. **Radius mode** — the radius and the end point are given. A positive radius selects the arc shorter than 180°, a negative radius the arc longer than 180°.

See 1.5 Motion Methods for the geometry of each mode.

Syntax

```
machine.arcYZ(r, dstY, dstZ);
machine.arcYZ(r, dstY, dstZ, revDir);
machine.arcYZ(r, dstY, dstZ, revDir, feedrate);
machine.arcYZ(cY, cZ, dstY, dstZ);
machine.arcYZ(cY, cZ, dstY, dstZ, revDir);
machine.arcYZ(cY, cZ, dstY, dstZ, revDir, feedrate);
```

Parameters

- **machine**: a **Machine** object.
- **r**: radius of the arc in radius mode (negative = arc greater than 180°).
- **cY**: center Y coordinate in center mode.
- **cZ**: center Z coordinate in center mode.
- **dstY**: target Y coordinate.
- **dstZ**: target Z coordinate.
- **revDir**: Direction of rotation: **false** = clockwise (default), **true** = counter-clockwise.
- **feedrate**: Feedrate. If omitted, the last recorded feedrate is used.

Returns

bool: **true** if the motion block was queued; **false** if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Whichever arc mode is used, a wrong parameter set may leave the target point off the circle, which results in unexpected machine motion.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
```

```

while (!Serial);
pinMode(EnablePin, OUTPUT);

// Reverse the direction of an axis if required.
// In this example the X and Y axes need to be reversed.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// PPU (pulses per unit) is a virtual length unit defined by your application.
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
    // Radius 10, counter-clockwise to (Y=10, Z=10).
    machine.arcYZ(10, 10, 10, true);

    // Radius 10, clockwise to (Y=0, Z=0).
    machine.arcYZ(10, 0, 0, false);

    // Center (Y=0, Z=10), counter-clockwise to (Y=10, Z=10).
    machine.arcYZ(0, 10, 10, 10, true);

    // Center (Y=0, Z=10), clockwise to (Y=0, Z=0).
    machine.arcYZ(0, 10, 0, 0, false);

    // Wait until the planned motion is finished.
    while (machine.isMoving());
}

```

arcXY_Theta()

Description

Moves along an arc in the XY plane using **angle mode**: the center point and the angle to sweep are given, and the end point is computed. The direction is judged looking from the positive direction of the Z axis: a positive angle is clockwise, a negative angle counter-clockwise.

See 1.5 Motion Methods for details.

Syntax

```
machine.arcXY_Theta(cX, cY, theta);
machine.arcXY_Theta(cX, cY, theta, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `cX`: center X coordinate.
- `cY`: center Y coordinate.
- `theta`: angle to rotate in radians; positive = clockwise, negative = counter-clockwise (e.g. `HALF_PI`, `-PI`).
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Whichever arc mode is used, a wrong parameter set may leave the target point off the circle, which results in unexpected machine motion.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
```

```
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Arc 90 degrees clockwise around center (X=0, Y=10).
  machine.arcXY_Theta(0, 10, HALF_PI, true);

  // Arc 90 degrees clockwise around center (X=0, Y=10).
  machine.arcXY_Theta(0, 10, HALF_PI, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

arcXZ_Theta()

Description

Moves along an arc in the XZ plane using **angle mode**: the center point and the angle to sweep are given, and the end point is computed. The direction is judged looking from the positive direction of the Y axis: a positive angle is clockwise, a negative angle counter-clockwise.

See 1.5 Motion Methods for details.

Syntax

```
machine.arcXZ_Theta(cX, cZ, theta);
machine.arcXZ_Theta(cX, cZ, theta, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `cX`: center X coordinate.
- `cZ`: center Z coordinate.
- `theta`: angle to rotate in radians; positive = clockwise, negative = counter-clockwise (e.g. `HALF_PI`, `-PI`).
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Whichever arc mode is used, a wrong parameter set may leave the target point off the circle, which results in unexpected machine motion.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
```

```
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Arc 90 degrees clockwise around center (X=0, Z=10).
  machine.arcXZ_Theta(0, 10, HALF_PI, true);

  // Arc 90 degrees clockwise around center (X=0, Z=10).
  machine.arcXZ_Theta(0, 10, HALF_PI, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

arcYZ_Theta()

Description

Moves along an arc in the YZ plane using **angle mode**: the center point and the angle to sweep are given, and the end point is computed. The direction is judged looking from the positive direction of the X axis: a positive angle is clockwise, a negative angle counter-clockwise.

See 1.5 Motion Methods for details.

Syntax

```
machine.arcYZ_Theta(cY, cZ, theta);
machine.arcYZ_Theta(cY, cZ, theta, feedrate);
```

Parameters

- **machine**: a Machine object.
- **cY**: center Y coordinate.
- **cZ**: center Z coordinate.
- **theta**: angle to rotate in radians; positive = clockwise, negative = counter-clockwise (e.g. HALF_PI, -PI).
- **feedrate**: Feedrate. If omitted, the last recorded feedrate is used.

Returns

bool: **true** if the motion block was queued; **false** if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Whichever arc mode is used, a wrong parameter set may leave the target point off the circle, which results in unexpected machine motion.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
```

```
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Arc 90 degrees clockwise around center (Y=0, Z=10).
  machine.arcYZ_Theta(0, 10, HALF_PI, true);

  // Arc 90 degrees clockwise around center (Y=0, Z=10).
  machine.arcYZ_Theta(0, 10, HALF_PI, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

circleXY()

Description

Draws one complete circle in the XY plane around the given center and returns to the starting point. The direction is judged looking from the positive direction of the Z axis.

See 1.5 Motion Methods for details.

Syntax

```
machine.circleXY(cX, cY);
machine.circleXY(cX, cY, revDir);
machine.circleXY(cX, cY, revDir, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `cX`: center X coordinate.
- `cY`: center Y coordinate.
- `revDir`: Direction of rotation: `false` = clockwise (default), `true` = counter-clockwise.
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
```

```
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Full circle counter-clockwise around center (X=0, Y=10).
  machine.circleXY(0, 10, true);

  // Full circle clockwise around center (X=0, Y=10).
  machine.circleXY(0, 10, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

circleXZ()

Description

Draws one complete circle in the XZ plane around the given center and returns to the starting point. The direction is judged looking from the positive direction of the Y axis.

See 1.5 Motion Methods for details.

Syntax

```
machine.circleXZ(cX, cZ);
machine.circleXZ(cX, cZ, revDir);
machine.circleXZ(cX, cZ, revDir, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `cX`: center X coordinate.
- `cZ`: center Z coordinate.
- `revDir`: Direction of rotation: `false` = clockwise (default), `true` = counter-clockwise.
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
```

```
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Full circle counter-clockwise around center (X=0, Z=10).
  machine.circleXZ(0, 10, true);

  // Full circle clockwise around center (X=0, Z=10).
  machine.circleXZ(0, 10, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

circleYZ()

Description

Draws one complete circle in the YZ plane around the given center and returns to the starting point. The direction is judged looking from the positive direction of the X axis.

See 1.5 Motion Methods for details.

Syntax

```
machine.circleYZ(cY, cZ);
machine.circleYZ(cY, cZ, revDir);
machine.circleYZ(cY, cZ, revDir, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `cY`: center Y coordinate.
- `cZ`: center Z coordinate.
- `revDir`: Direction of rotation: `false` = clockwise (default), `true` = counter-clockwise.
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
```

```
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
    // Full circle counter-clockwise around center (Y=0, Z=10).
    machine.circleYZ(0, 10, true);

    // Full circle clockwise around center (Y=0, Z=10).
    machine.circleYZ(0, 10, false);

    // Wait until the planned motion is finished.
    while (machine.isMoving());
}
```

helicalXY()

Description

Moves along a helix: a circular motion of `theta` radians in the XY plane around the given center, combined with a linear move of the Z axis to `dstZ`. The direction is judged looking from the positive direction of the Z axis: a positive angle is clockwise, a negative angle counter-clockwise. See 1.5 Motion Methods for details.

Syntax

```
machine.helicalXY(cX, cY, dstZ, theta);
machine.helicalXY(cX, cY, dstZ, theta, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `cX`: center X coordinate.
- `cY`: center Y coordinate.
- `dstZ`: target Z coordinate reached at the end of the helix.
- `theta`: total angle to rotate in radians (e.g. $4 * \text{PI}$ = two turns); positive = clockwise, negative = counter-clockwise.
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
```

```
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Helix TWO_PI * 2 rad clockwise around center (X=0, Y=10) while Z moves to 10.
  machine.helicalXY(0, 10, 10, TWO_PI * 2, true);

  // Helix TWO_PI * 2 rad clockwise around center (X=0, Y=10) while Z moves to 0.
  machine.helicalXY(0, 10, 0, TWO_PI * 2, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

helicalXZ()

Description

Moves along a helix: a circular motion of `theta` radians in the XZ plane around the given center, combined with a linear move of the Y axis to `dstY`. The direction is judged looking from the positive direction of the Y axis: a positive angle is clockwise, a negative angle counter-clockwise. See 1.5 Motion Methods for details.

Syntax

```
machine.helicalXZ(cX, cZ, dstY, theta);
machine.helicalXZ(cX, cZ, dstY, theta, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `cX`: center X coordinate.
- `cZ`: center Z coordinate.
- `dstY`: target Y coordinate reached at the end of the helix.
- `theta`: total angle to rotate in radians (e.g. $4 * \text{PI}$ = two turns); positive = clockwise, negative = counter-clockwise.
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
```

```
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Helix TWO_PI * 2 rad clockwise around center (X=0, Z=10) while Y moves to 10.
  machine.helicalXZ(0, 10, 10, TWO_PI * 2, true);

  // Helix TWO_PI * 2 rad clockwise around center (X=0, Z=10) while Y moves to 0.
  machine.helicalXZ(0, 10, 0, TWO_PI * 2, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

helicalYZ()

Description

Moves along a helix: a circular motion of `theta` radians in the YZ plane around the given center, combined with a linear move of the X axis to `dstX`. The direction is judged looking from the positive direction of the X axis: a positive angle is clockwise, a negative angle counter-clockwise. See 1.5 Motion Methods for details.

Syntax

```
machine.helicalYZ(cY, cZ, dstX, theta);
machine.helicalYZ(cY, cZ, dstX, theta, feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `cY`: center Y coordinate.
- `cZ`: center Z coordinate.
- `dstX`: target X coordinate reached at the end of the helix.
- `theta`: total angle to rotate in radians (e.g. $4 * \text{PI}$ = two turns); positive = clockwise, negative = counter-clockwise.
- `feedrate`: Feedrate. If omitted, the last recorded feedrate is used.

Returns

`bool`: `true` if the motion block was queued; `false` if the command is not allowed (machine off, in JOG/MPG mode, emergency-stop state, or command buffer full).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
```

```
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Helix TWO_PI * 2 rad clockwise around center (Y=0, Z=10) while X moves to 10.
  machine.helicalYZ(0, 10, 10, TWO_PI * 2, true);

  // Helix TWO_PI * 2 rad clockwise around center (Y=0, Z=10) while X moves to 0.
  machine.helicalYZ(0, 10, 0, TWO_PI * 2, false);

  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

2.1.3 Write Configuration

Machine set-up. All `config_*` methods must be called before `machineOn()`.

Functions:

- [`config\_MaxVelocity\(\)`](#)
- [`config\_MaxAcc\(\)`](#)
- [`config\_PosLimit\(\)`](#)
- [`config\_PPU\(\)`](#)
- [`config\_ReverseDirection\(\)`](#)
- [`config\_MaxPulseSpeed\(\)`](#)
- [`config\_PulseMode\(\)`](#)
- [`config\_HomePins\(\)`](#)
- [`setDefaultFeedrate\(\)`](#)
- [`setHomeSpeed\(\)`](#)
- [`setRelative\(\)`](#)
- [`setAbsolute\(\)`](#)
- [`enableSoftLimit\(\)`](#)
- [`disableSoftLimit\(\)`](#)

config_MaxVelocity()

Description

Sets the maximum speed of the specified axis.

Syntax

```
machine.config_MaxVelocity(axis, max_feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.
- `max_feedrate`: maximum feedrate of the axis, in mm/s (user units per second).

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // Limit every axis to 50 mm/s and the machine acceleration to 500 mm/s^2.
  machine.config_MaxVelocity(AXIS_X, 50.0);
  machine.config_MaxVelocity(AXIS_Y, 50.0);
  machine.config_MaxVelocity(AXIS_Z, 10.0);
```

```
machine.config_MaxAcc(500.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
    // Motion control code...
}
```

config_MaxAcc()

Description

Sets the maximum acceleration used by the planner for the machine.

Syntax

```
machine.config_MaxAcc(max_acc);
```

Parameters

- `machine`: a `Machine` object.
- `max_acc`: maximum feed acceleration, in mm/s² (user units per second squared).

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // Machine acceleration limit.
  machine.config_MaxAcc(500.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
```

```
// Enable the stepper drivers.  
digitalWrite(EnablePin, LOW);  
}  
  
void loop() {  
  // Motion control code...  
}
```

config_PosLimit()

Description

Sets the software travel limits of the specified axis. The limits take effect after `enableSoftLimit()` is called.

Syntax

```
machine.config_PosLimit(axis, min, max);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.
- `min`: minimum position of the axis; the unit follows the PPU setting (normally mm).
- `max`: maximum position of the axis; the unit follows the PPU setting (normally mm).

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

  while (!Serial);
  pinMode(EnablePin, OUTPUT);
  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);
  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
  // Set the software limits of the machine.
  machine.config_PosLimit(AXIS_X, 0, 300);
  machine.config_PosLimit(AXIS_Y, 0, 200);
  machine.config_PosLimit(AXIS_Z, 0, 300);
  // Set the limit-switch pins that define the home point.
```

```
machine.config_HomePins(2, 7, 8);
// The machine must be turned on before it can be controlled.
machine.machineOn();
// Set the default feedrate.
machine.setDefaultFeedrate(400);
// Enable the software limits.
machine.enableSoftLimit();
// Set the homing feedrate.
machine.setHomeSpeed(1000, 1000, 200);
// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
// Return to the home point defined by the limit switches.
machine.home();

}

void loop() {
  // Motion control code...
}
```

config_PPU()

Description

Sets the Pulses per Unit (PPU) of the specified axis. The PPU defines how many pulses (or encoder counts on a Custom Machine) correspond to one unit of travel, e.g. 80 pulses/mm. All positions and feedrates of the machine are expressed in this unit.

Use `computePPU_Belt()` or `computePPU_LeadScrew()` to derive the PPU from the mechanical parameters.

Syntax

```
machine.config_PPU(axis, pulses_per_unit);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.
- `pulses_per_unit`: number of pulses to output for one unit of travel on the axis, e.g. pulses/mm.

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

  while (!Serial);
  pinMode(EnablePin, OUTPUT);
  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);
  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
  // Set the software limits of the machine.
  machine.config_PosLimit(AXIS_X, 0, 300);
```

```
machine.config_PosLimit(AXIS_Y, 0, 200);
machine.config_PosLimit(AXIS_Z, 0, 300);
// Set the limit-switch pins that define the home point.
machine.config_HomePins(2, 7, 8);
// The machine must be turned on before it can be controlled.
machine.machineOn();
// Set the default feedrate.
machine.setDefaultFeedrate(400);
// Enable the software limits.
machine.enableSoftLimit();
// Set the homing feedrate.
machine.setHomeSpeed(1000, 1000, 200);
// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
// Return to the home point defined by the limit switches.
machine.home();

}

void loop() {
  // Motion control code...
}
```

config_ReverseDirection()

Description

Reverses the direction of motion of the specified axis.

Syntax

```
machine.config_ReverseDirection(axis);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

  while (!Serial);
  pinMode(EnablePin, OUTPUT);
  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);
  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
  // Set the software limits of the machine.
  machine.config_PosLimit(AXIS_X, 0, 300);
  machine.config_PosLimit(AXIS_Y, 0, 200);
  machine.config_PosLimit(AXIS_Z, 0, 300);
  // Set the limit-switch pins that define the home point.
  machine.config_HomePins(2, 7, 8);
  // The machine must be turned on before it can be controlled.
  machine.machineOn();
}
```

```
// Set the default feedrate.
machine.setDefaultFeedrate(400);
// Enable the software limits.
machine.enableSoftLimit();
// Set the homing feedrate.
machine.setHomeSpeed(1000, 1000, 200);
// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
// Return to the home point defined by the limit switches.
machine.home();

}

void loop() {
  // Motion control code...
}
```

config_MaxPulseSpeed()

Description

Sets the maximum pulse output frequency of the machine.

This method only applies to local machines (MACHINE_0~MACHINE_2).

Syntax

```
machine.config_MaxPulseSpeed(pfHz);
```

Parameters

- `machine`: a `Machine` object.
- `pfHz`: maximum pulse output frequency, in pulses per second (Hz).

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // Limit the pulse output to 1 MHz.
  machine.config_MaxPulseSpeed(1000000);

  // The machine must be turned on before it can be controlled.
```

```
machine.machineOn();

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
    // Motion control code...
}
```

config_PulseMode()

Description

Sets the pulse output mode (servo / stepper driver signal type) of the specified axis.

Syntax

```
machine.config_PulseMode(axis, mode);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.
- `mode`: output mode of the axis:
 - `MODE_STEP_DIR` (0): Step (pulse) and Direction signals (default).
 - `MODE_CWCCW` (1): separate clockwise and counter-clockwise pulse signals.
 - `MODE_AB_PHASE` (2): A/B phase quadrature signals.
 - `MODE_CAPTURE` (3): position capture mode.
 - `MODE_SSI` (4): Synchronous Serial Interface (continuous mode, no interrupt).

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
```

```
// The X-axis driver expects CW / CCW pulses instead of Step / Dir.  
machine.config_PulseMode(AXIS_X, MODE_CWCCW);  
  
// The machine must be turned on before it can be controlled.  
machine.machineOn();  
  
// Enable the stepper drivers.  
digitalWrite(EnablePin, LOW);  
}  
  
void loop() {  
  // Motion control code...  
}
```

config_HomePins()

Description

Sets the pins of the hardware limit switches used to define the home point of the machine. `home()` moves each axis in the negative direction until the corresponding switch is triggered.

Syntax

```
machine.config_HomePins(xLimit, yLimit, zLimit);
```

Parameters

- `machine`: a `Machine` object.
- `xLimit`: pin of the X-axis limit switch.
- `yLimit`: pin of the Y-axis limit switch.
- `zLimit`: pin of the Z-axis limit switch.

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

  while (!Serial);
  pinMode(EnablePin, OUTPUT);
  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);
  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
  // Set the software limits of the machine.
  machine.config_PosLimit(AXIS_X, 0, 300);
  machine.config_PosLimit(AXIS_Y, 0, 200);
  machine.config_PosLimit(AXIS_Z, 0, 300);
  // Set the limit-switch pins that define the home point.
```

```
machine.config_HomePins(2, 7, 8);
// The machine must be turned on before it can be controlled.
machine.machineOn();
// Set the default feedrate.
machine.setDefaultFeedrate(400);
// Enable the software limits.
machine.enableSoftLimit();
// Set the homing feedrate.
machine.setHomeSpeed(1000, 1000, 200);
// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
// Return to the home point defined by the limit switches.
machine.home();

}

void loop() {
  // Motion control code...
}
```

setDefaultFeedrate()

Description

Sets the default feedrate of the machine. It is used by motion methods that are called without a `feedrate` argument, until another feedrate is recorded.

Syntax

```
machine.setDefaultFeedrate(feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `feedrate`: default feedrate for every axis; only positive values are accepted.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

    while (!Serial);
    pinMode(EnablePin, OUTPUT);
    // Reverse the direction of an axis if required.
    machine.config_ReverseDirection(AXIS_X);
    machine.config_ReverseDirection(AXIS_Y);
    // PPU (pulses per unit) is a virtual length unit defined by your application.
    // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
    machine.config_PPU(AXIS_X, 80.0);
    machine.config_PPU(AXIS_Y, 80.0);
    machine.config_PPU(AXIS_Z, 1600.0);
    // Set the software limits of the machine.
    machine.config_PosLimit(AXIS_X, 0, 300);
    machine.config_PosLimit(AXIS_Y, 0, 200);
    machine.config_PosLimit(AXIS_Z, 0, 300);
    // Set the limit-switch pins that define the home point.
    machine.config_HomePins(2, 7, 8);
    // The machine must be turned on before it can be controlled.
    machine.machineOn();
    // Set the default feedrate.
```

```
machine.setDefaultFeedrate(400);  
// Enable the software limits.  
machine.enableSoftLimit();  
// Set the homing feedrate.  
machine.setHomeSpeed(1000, 1000, 200);  
// Enable the stepper drivers.  
digitalWrite(EnablePin, LOW);  
// Return to the home point defined by the limit switches.  
machine.home();  
  
}  
  
void loop() {  
  // Motion control code...  
}
```

setHomeSpeed()

Description

Sets the feedrate used by `home()`. A single value applies to all three axes; the three-argument form sets each axis individually.

Syntax

```
machine.setHomeSpeed(feedrate);
```

```
machine.setHomeSpeed(xFeedrate, yFeedrate, zFeedrate);
```

Parameters

- `machine`: a `Machine` object.
- `feedrate`: homing feedrate for the X, Y and Z axes.
- `xFeedrate`: homing feedrate of the X axis.
- `yFeedrate`: homing feedrate of the Y axis.
- `zFeedrate`: homing feedrate of the Z axis.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

  while (!Serial);
  pinMode(EnablePin, OUTPUT);
  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);
  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
  // Set the software limits of the machine.
  machine.config_PosLimit(AXIS_X, 0, 300);
  machine.config_PosLimit(AXIS_Y, 0, 200);
  machine.config_PosLimit(AXIS_Z, 0, 300);
  // Set the limit-switch pins that define the home point.
```

```
machine.config_HomePins(2, 7, 8);
// The machine must be turned on before it can be controlled.
machine.machineOn();
// Set the default feedrate.
machine.setDefaultFeedrate(400);
// Enable the software limits.
machine.enableSoftLimit();
// Set the homing feedrate.
machine.setHomeSpeed(1000, 1000, 200);
// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
// Return to the home point defined by the limit switches.
machine.home();

}

void loop() {
  // Motion control code...
}
```

setRelative()

Description

Switches the machine to relative (incremental) positioning: the target coordinates of the motion methods are interpreted as increments from the current position. Equivalent to G-code **G91**.

Syntax

```
machine.setRelative();
```

Parameters

- **machine**: a **Machine** object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);
  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);
  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
  // The machine must be turned on before it can be controlled.
  machine.machineOn();
  // Set the default feedrate.
  machine.setDefaultFeedrate(600);
  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
  // Plan motion in relative increments.
  machine.setRelative();
  // Move to the center position using relative increments from the current position.
  machine.line(0,0,10,400);
  machine.line(170,100,0,800);
```

```
machine.line(0,0,-10,400);  
}  
  
void loop() {  
    // Move to the relative position; no feedrate given, so the default feedrate is used.  
    machine.line(10,0,0);  
    machine.line(0,10,0);  
    machine.line(-10,0,0);  
    machine.line(0,-10,0);  
    // Wait until the planned motion is finished.  
    while (machine.isMoving());  
}
```

setAbsolute()

Description

Switches the machine to absolute positioning (default): the target coordinates of the motion methods are absolute positions relative to the home point. Equivalent to G-code `G90`.

Syntax

```
machine.setAbsolute();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);
// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);
  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);
  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
  // The machine must be turned on before it can be controlled.
  machine.machineOn();
  // Set the default feedrate.
  machine.setDefaultFeedrate(600);
  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
  // Plan motion in absolute coordinates relative to the home point.
  // Without a configured home point, the power-on position is used as home.
  machine.setAbsolute();
  // Move to the center position using absolute coordinates.
  machine.line(170,100,10,400);
```

```
machine.line(170,100,10,800);  
machine.line(170,100,0,400);  
}  
  
void loop() {  
  // Move to the absolute position; no feedrate given, so the default feedrate is used.  
  machine.line(170,100,0);  
  machine.line(180,100,0);  
  machine.line(180,110,0);  
  machine.line(170,110,0);  
  // Wait until the planned motion is finished.  
  while (machine.isMoving());  
}
```

enableSoftLimit()

Description

Enables the software travel limits set with `config_PosLimit()`. Call it after `machineOn()`.

Syntax

```
machine.enableSoftLimit();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // Software travel limits of each axis.
  machine.config_PosLimit(AXIS_X, 0, 300);
  machine.config_PosLimit(AXIS_Y, 0, 200);
  machine.config_PosLimit(AXIS_Z, 0, 300);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
```

```
// Enable the software limits to make control safer.  
machine.enableSoftLimit();  
  
// Enable the stepper drivers.  
digitalWrite(EnablePin, LOW);  
}  
  
void loop() {  
  // Motion control code...  
}
```

disableSoftLimit()

Description

Disables the software travel limits.

Syntax

```
machine.disableSoftLimit();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  machine.config_PosLimit(AXIS_X, 0, 300);
  machine.config_PosLimit(AXIS_Y, 0, 200);
  machine.config_PosLimit(AXIS_Z, 0, 300);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  machine.enableSoftLimit();
```

```
// Enable the stepper drivers.  
digitalWrite(EnablePin, LOW);  
}  
  
void loop() {  
  // Temporarily disable the software limits, e.g. for a manual set-up move.  
  machine.disableSoftLimit();  
  machine.line(-5, -5, 0, 200);  
  while (machine.isMoving());  
  machine.enableSoftLimit();  
  while (1);  
}
```

2.1.4 Read Configuration

Read back configuration and run-time status.

Functions:

- [getMaxVelocity\(\)](#)
- [getMaxAcc\(\)](#)
- [getPPU\(\)](#)
- [getActualPos\(\)](#)
- [getPulseMode\(\)](#)
- [getCmdCount\(\)](#)
- [getFreeCmdCount\(\)](#)
- [isDirectionReversed\(\)](#)
- [isMoving\(\)](#)
- [isValid\(\)](#)
- [isCmdBufferFull\(\)](#)

getMaxVelocity()

Description

Returns the maximum speed limit of the specified axis.

Syntax

```
machine.getMaxVelocity(axis);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.

Returns

`double`: maximum speed of the axis.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}
```

```
void loop() {  
  Serial.print("Max velocity X = ");  
  Serial.println(machine.getMaxVelocity(AXIS_X));  
  delay(1000);  
}
```

getMaxAcc()

Description

Returns the maximum acceleration limit of the machine.

Syntax

```
machine.getMaxAcc();
```

Parameters

- `machine`: a `Machine` object.

Returns

`double`: maximum acceleration of the machine.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}

void loop() {
```

```
Serial.print("Max acceleration = ");  
Serial.println(machine.getMaxAcc());  
delay(1000);  
}
```

getPPU()

Description

Returns the PPU (pulses per unit) of the specified axis.

Syntax

```
machine.getPPU(axis);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.

Returns

`double`: PPU value of the axis.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}
```

```
void loop() {  
  Serial.print("PPU X = ");  
  Serial.println(machine.getPPU(Axis_X));  
  delay(1000);  
}
```

getActualPos()

Description

Returns the current position of the machine in the workspace, either for one axis or for all three axes at once through reference arguments.

Syntax

```
machine.getActualPos(axis);
```

```
machine.getActualPos(x, y, z);
```

Parameters

- **machine**: a **Machine** object.
- **axis**: the axis to configure: **AXIS_X**, **AXIS_Y** or **AXIS_Z**.
- **x**: reference to a **double** that receives the X position.
- **y**: reference to a **double** that receives the Y position.
- **z**: reference to a **double** that receives the Z position.

Returns

double: position of the requested axis (single-axis form). The three-axis form returns nothing and writes the positions into **x**, **y**, **z**.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
```

```
machine.machineOn();

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
    double x, y, z;
    machine.line(10, 10, 0, 400);
    while (machine.isMoving()) {
        machine.getActualPos(x, y, z);
        Serial.print("X:"); Serial.print(x);
        Serial.print(" Y:"); Serial.print(y);
        Serial.print(" Z:"); Serial.println(z);
        delay(100);
    }
    machine.line(0, 0, 0, 400);
    while (machine.isMoving());
}
```

getPulseMode()

Description

Returns the pulse output mode of the specified axis.

Syntax

```
machine.getPulseMode(axis);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.

Returns

`pulseMode_t`: `MODE_STEP_DIR` (0), `MODE_CWCCW` (1), `MODE_AB_PHASE` (2), `MODE_CAPTURE` (3) or `MODE_SSI` (4).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}
```

```
void loop() {  
  if (machine.getPulseMode(AXIS_X) == MODE_STEP_DIR)  
    Serial.println("X axis: Step / Dir mode");  
  delay(1000);  
}
```

getCmdCount()

Description

Returns the number of motion blocks currently queued in the command buffer. The buffer holds 1024 blocks.

Syntax

```
machine.getCmdCount();
```

Parameters

- `machine`: a `Machine` object.

Returns

`int`: number of used entries in the command buffer.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}
```

```
void loop() {  
  for (int i = 0; i < 20; i++)  
    machine.line(i, i, 0, 400);  
  Serial.print("Queued blocks = ");  
  Serial.println(machine.getCmdCount());  
  while (machine.isMoving());  
}
```

getFreeCmdCount()

Description

Returns the number of free entries in the command buffer. The buffer holds 1024 blocks.

Syntax

```
machine.getFreeCmdCount();
```

Parameters

- `machine`: a `Machine` object.

Returns

`int`: number of free entries in the command buffer.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}

void loop() {
```

```
if (machine.getFreeCmdCount() > 0)
    machine.gcode("G1 X10 Y10 F400");
while (machine.isMoving());
}
```

isDirectionReversed()

Description

Reports whether the specified axis has been configured to move in the reverse direction.

Syntax

```
machine.isDirectionReversed(axis);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.

Returns

`bool`: `true` if the axis is reversed; `false` if it is not reversed, or if the machine does not exist.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}
```

```
void loop() {  
  Serial.print("X reversed: ");  
  Serial.println(machine.isDirectionReversed(Axis_X) ? "yes" : "no");  
  delay(1000);  
}
```

isMoving()

Description

Reports whether the machine is currently moving or still has motion blocks to execute.

Syntax

```
machine.isMoving();
```

Parameters

- `machine`: a `Machine` object.

Returns

`bool`: `true` if the machine is moving; `false` if it is idle, or if the machine does not exist.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}

void loop() {
```

```
machine.line(10, 10, 10, 400);  
machine.line(0, 0, 0, 400);  
// Wait until the planned motion is finished.  
while (machine.isMoving());  
Serial.println("Done");  
delay(1000);  
}
```

isValid()

Description

Reports whether the `Machine` object exists and was created successfully. Creating a second object on an axis group that is already in use yields an invalid machine.

Syntax

```
machine.isValid();
```

Parameters

- `machine`: a `Machine` object.

Returns

`bool`: `true` if the machine exists and was created successfully; `false` otherwise.

Example

```
#include "Motion86.h"

Machine machine(0);

void setup() {
  while (!Serial);
  if (!machine.isValid()) {
    Serial.println("Machine 0 is not available.");
    while (1);
  }
  machine.config_PPU(Axis_X, 80.0);
  machine.config_PPU(Axis_Y, 80.0);
  machine.config_PPU(Axis_Z, 1600.0);
  machine.machineOn();
}

void loop() {
}
```

isCmdBufferFull()

Description

Reports whether the command buffer (1024 blocks) is full. Poll it before queuing further commands when streaming a G-code file.

Syntax

```
machine.isCmdBufferFull();
```

Parameters

- `machine`: a `Machine` object.

Returns

`bool`: `true` if the buffer is full; `false` if there is space left.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}
```

```
void loop() {  
  // Queue as many blocks as the buffer accepts.  
  while (!machine.isCmdBufferFull())  
    machine.gcode("G91 G1 X1 F400");  
  while (machine.isMoving());  
}
```

2.1.5 JOG / MPG Control

Manual operation with jog switches or an MPG hand wheel.

Functions:

- [beginJog\(\)](#)
- [endJog\(\)](#)
- [setJogSpeed\(\)](#)
- [setJogOffset\(\)](#)
- [setJogPins\(\)](#)
- [beginMpg\(\)](#)
- [endMpg\(\)](#)
- [setMpgOn\(\)](#)
- [setMpgOff\(\)](#)
- [setMpgSpeed\(\)](#)
- [setMpgAxis\(\)](#)
- [setMpgRatio\(\)](#)
- [getJogPos\(\)](#)

beginJog()

Description

Enters JOG mode. In JOG mode the machine accepts no other motion command; it is driven only by the switches connected to the jog pins.

Two behaviours are available:

Jog mode — the axis moves while the switch is held and stops when it is released.

incJog mode — every press moves the axis by a fixed distance set with `setJogOffset()`.

The form without pins uses the pins assigned earlier with `setJogPins()` or, on a Custom Machine, the inputs provided by the jog callback / plugin (e.g. `EcatMachine::attach_EcatJogPins_X()`).

Syntax

```
machine.beginJog();
machine.beginJog(incJog);
machine.beginJog(pX, nX, pY, nY, pZ, nZ);
machine.beginJog(pX, nX, pY, nY, pZ, nZ, incJog);
machine.beginJog(pX, nX, pY, nY, pZ, nZ, incJog, polarity);
```

Parameters

- `machine`: a `Machine` object.
- `pX`: pin of the X-axis positive-direction switch.
- `nX`: pin of the X-axis negative-direction switch.
- `pY`: pin of the Y-axis positive-direction switch.
- `nY`: pin of the Y-axis negative-direction switch.
- `pZ`: pin of the Z-axis positive-direction switch.
- `nZ`: pin of the Z-axis negative-direction switch.
- `incJog`: `true` to enable incJog mode; default `false`.
- `polarity`: active level of the switches: `ACTIVE_HIGH` (default) or `ACTIVE_LOW`.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);
```

```

// Reverse the direction of an axis if required.
// In this example the X and Y axes need to be reversed.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// PPU (pulses per unit) is a virtual length unit defined by your application.
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();

// Put the machine in Jog mode, triggered by the buttons wired to these pins.
machine.beginJog(27, 28, 32, 3, 16, 17, false);

// If incJog mode is used, set the jog offset to define how many millimeters to move per
click.
// machine.beginJog(27, 28, 32, 3, 16, 17, true);
// machine.setJogOffset(1.0);

// Feedrate in Jog mode.
machine.setJogSpeed(600);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Jog position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}

```

endJog()

Description

Leaves JOG mode and returns to normal motion control.

Syntax

```
machine.endJog();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Jog for 10 seconds, then return to normal control.
  machine.beginJog(27, 28, 32, 3, 16, 17, false);
  machine.setJogSpeed(600);
  delay(10000);
  machine.endJog();
```

```
// Enable the stepper drivers.  
digitalWrite(EnablePin, LOW);  
}  
  
void loop() {  
  // Motion control code...  
}
```

setJogSpeed()

Description

Sets the feedrate used in JOG / incJog mode.

Syntax

```
machine.setJogSpeed(feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `feedrate`: jog feedrate.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Put the machine in Jog mode, triggered by the buttons wired to these pins.
  machine.beginJog(27, 28, 32, 3, 16, 17, false);
```

```
// If incJog mode is used, set the jog offset to define how many millimeters to move per
click.
// machine.beginJog(27, 28, 32, 3, 16, 17, true);
// machine.setJogOffset(1.0);

// Feedrate in Jog mode.
machine.setJogSpeed(600);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Jog position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}
```

setJogOffset()

Description

Sets the distance moved per switch press in incJog mode.

Syntax

```
machine.setJogOffset(incPerMM);
```

Parameters

- `machine`: a `Machine` object.
- `incPerMM`: distance to move per trigger, in mm (user units).

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Put the machine in Jog mode, triggered by the buttons wired to these pins.
  // machine.beginJog(27, 28, 32, 3, 16, 17, false);
```

```
// If incJog mode is used, set the jog offset to define how many millimeters to move per
click.
machine.beginJog(27, 28, 32, 3, 16, 17, true);
machine.setJogOffset(1.0);

// Feedrate in Jog mode.
machine.setJogSpeed(600);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Jog position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}
```

setJogPins()

Description

Assigns the jog switch pins of each axis direction without entering JOG mode. Use it together with `beginJog()` / `beginJog(incJog)`.

Syntax

```
machine.setJogPins(pX, nX, pY, nY, pZ, nZ);
machine.setJogPins(pX, nX, pY, nY, pZ, nZ, polarity);
```

Parameters

- `machine`: a `Machine` object.
- `pX`: pin of the X-axis positive-direction switch.
- `nX`: pin of the X-axis negative-direction switch.
- `pY`: pin of the Y-axis positive-direction switch.
- `nY`: pin of the Y-axis negative-direction switch.
- `pZ`: pin of the Z-axis positive-direction switch.
- `nZ`: pin of the Z-axis negative-direction switch.
- `polarity`: active level of the switches: `ACTIVE_HIGH` (default) or `ACTIVE_LOW`.

Returns

None.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
```

```
// The machine must be turned on before it can be controlled.
machine.machineOn();

// Assign active-low jog buttons, then start continuous jog mode.
machine.setJogPins(27, 28, 32, 3, 16, 17, ACTIVE_LOW);
machine.setJogSpeed(600);
machine.beginJog();

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Jog position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}
```

beginMpg()

Description

Enters MPG (hand-wheel) mode. In MPG mode the machine accepts no other motion command; it follows the change of the encoder count on the axis selected with `setMpgAxis()`.

`beginMpg(encoder)` uses a local 86Duino `Encoder` object (e.g. `Enc3`). `beginMpg()` without argument uses the MPG source provided by a Custom Machine – the MPG callback (`config_MpgController_Callback()`) or a plugin such as `EcatMachine::attach_EcatMpg()`.

Syntax

```
machine.beginMpg();
machine.beginMpg(encoder);
```

Parameters

- `machine`: a `Machine` object.
- `encoder`: an `Encoder` object (`Enc0~Enc3`) connected to the hand wheel.

Returns

None.

Example

```
#include "Motion86.h"
#include "Encoder.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

    while (!Serial);
    // In this example the hand wheel is connected to Enc3.
    Enc3.begin(MODE_AB_PHASE);
    pinMode(EnablePin, OUTPUT);

    // Reverse the direction of an axis if required.
    // In this example the X and Y axes need to be reversed.
    machine.config_ReverseDirection(AXIS_X);
    machine.config_ReverseDirection(AXIS_Y);

    // PPU (pulses per unit) is a virtual length unit defined by your application.
    // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
    machine.config_PPU(AXIS_X, 80.0);
```

```

machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();

// Use Enc3 as the MPG source.
machine.beginMpg(Enc3);

// Select the axis to be driven by the MPG hand wheel.
machine.setMpgAxis(AXIS_X);
// machine.setMpgAxis(AXIS_Y);
// machine.setMpgAxis(AXIS_Z);

// Feedrate in MPG mode.
machine.setMpgSpeed(600);

// MPG ratio is the ratio between hand-wheel counts and machine travel; it must be greater
than 0.
machine.setMpgRatio(1);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Mpg position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}

```

endMpg()

Description

Leaves MPG mode and returns to normal motion control.

Syntax

```
machine.endMpg();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"
#include "Encoder.h"

Machine machine(0);
int EnablePin = 12;

void setup() {
  Enc3.begin(MODE_AB_PHASE);
  pinMode(EnablePin, OUTPUT);
  machine.config_PPU(Axis_X, 80.0);
  machine.config_PPU(Axis_Y, 80.0);
  machine.config_PPU(Axis_Z, 1600.0);
  machine.machineOn();
  digitalWrite(EnablePin, LOW);

  // Hand-wheel control on the X axis for 10 seconds.
  machine.beginMpg(Enc3);
  machine.setMpgAxis(Axis_X);
  machine.setMpgSpeed(600);
  machine.setMpgRatio(1);
  delay(10000);
  machine.endMpg();

  // Back to normal control.
  machine.line(0, 0, 0, 400);
}

void loop() {
}
```

setMpgOn()

Description

Re-enables hand-wheel following while the machine stays in MPG mode. Use it together with `setMpgOff()` to pause and resume the hand wheel without leaving MPG mode, for example while changing the target axis.

Syntax

```
machine.setMpgOn();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
#include "Motion86.h"
#include "Encoder.h"

Machine machine(0);
int EnablePin = 12;

void setup() {
  while (!Serial);
  Enc3.begin(MODE_AB_PHASE);
  pinMode(EnablePin, OUTPUT);
  machine.config_PPU(Axis_X, 80.0);
  machine.config_PPU(Axis_Y, 80.0);
  machine.config_PPU(Axis_Z, 1600.0);
  machine.machineOn();
  digitalWrite(EnablePin, LOW);

  machine.beginMpg(Enc3);
  machine.setMpgAxis(Axis_X);
  machine.setMpgSpeed(600);
  machine.setMpgRatio(1);
}

void loop() {
  // Drive X for 5 seconds, pause the hand wheel, switch to Y, resume.
  delay(5000);
  machine.setMpgOff();
  machine.setMpgAxis(Axis_Y);
  machine.setMpgOn();
}
```

```
delay(5000);  
machine.setMpgOff();  
machine.setMpgAxis(Axis_X);  
machine.setMpgOn();  
}
```

setMpgOff()

Description

Temporarily disables hand-wheel following while the machine stays in MPG mode. Hand-wheel movement is ignored until `setMpgOn()` is called.

Syntax

```
machine.setMpgOff();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

setMpgSpeed()

Description

Sets the feedrate used in MPG mode.

Syntax

```
machine.setMpgSpeed(feedrate);
```

Parameters

- `machine`: a `Machine` object.
- `feedrate`: MPG feedrate.

Returns

None.

Example

```
#include "Motion86.h"
#include "Encoder.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

    while (!Serial);

    // In this example the hand wheel is connected to Enc3.
    Enc3.begin(MODE_AB_PHASE);
    pinMode(EnablePin, OUTPUT);

    // Reverse the direction of an axis if required.
    // In this example the X and Y axes need to be reversed.
    machine.config_ReverseDirection(AXIS_X);
    machine.config_ReverseDirection(AXIS_Y);

    // PPU (pulses per unit) is a virtual length unit defined by your application.
    // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
    machine.config_PPU(AXIS_X, 80.0);
    machine.config_PPU(AXIS_Y, 80.0);
    machine.config_PPU(AXIS_Z, 1600.0);

    // The machine must be turned on before it can be controlled.
    machine.machineOn();
}
```

```

// Use Enc3 as the MPG source.
machine.beginMpg(Enc3);

// Select the axis to be driven by the MPG hand wheel.
machine.setMpgAxis(Axis_X);
// machine.setMpgAxis(Axis_Y);
// machine.setMpgAxis(Axis_Z);

// Feedrate in MPG mode.
machine.setMpgSpeed(600);

// MPG ratio is the ratio between hand-wheel counts and machine travel; it must be greater
than 0.
machine.setMpgRatio(1);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Mpg position = ");
  Serial.print(machine.getJogPos(Axis_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(Axis_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(Axis_Z));
  delay(500);
}

```

setMpgAxis()

Description

Selects the axis driven by the hand wheel in MPG mode.

Syntax

```
machine.setMpgAxis(axis);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the output axis: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.

Returns

None.

Example

```
#include "Motion86.h"
#include "Encoder.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

    while (!Serial);

    // In this example the hand wheel is connected to Enc3.
    Enc3.begin(MODE_AB_PHASE);
    pinMode(EnablePin, OUTPUT);

    // Reverse the direction of an axis if required.
    // In this example the X and Y axes need to be reversed.
    machine.config_ReverseDirection(AXIS_X);
    machine.config_ReverseDirection(AXIS_Y);

    // PPU (pulses per unit) is a virtual length unit defined by your application.
    // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
    machine.config_PPU(AXIS_X, 80.0);
    machine.config_PPU(AXIS_Y, 80.0);
    machine.config_PPU(AXIS_Z, 1600.0);

    // The machine must be turned on before it can be controlled.
    machine.machineOn();
}
```

```

// Use Enc3 as the MPG source.
machine.beginMpg(Enc3);

// Select the axis to be driven by the MPG hand wheel.
machine.setMpgAxis(AXIS_X);
// machine.setMpgAxis(AXIS_Y);
// machine.setMpgAxis(AXIS_Z);

// Feedrate in MPG mode.
machine.setMpgSpeed(600);

// MPG ratio is the ratio between hand-wheel counts and machine travel; it must be greater
than 0.
machine.setMpgRatio(1);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Mpg position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}

```

setMpgRatio()

Description

Sets the multiplier between the encoder count change and the travelled distance in MPG mode.

Syntax

```
machine.setMpgRatio(rate);
```

Parameters

- `machine`: a `Machine` object.
- `rate`: ratio between encoder counts and distance; must be greater than 0.

Returns

None.

Example

```
#include "Motion86.h"
#include "Encoder.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

    while (!Serial);

    // In this example the hand wheel is connected to Enc3.
    Enc3.begin(MODE_AB_PHASE);
    pinMode(EnablePin, OUTPUT);

    // Reverse the direction of an axis if required.
    // In this example the X and Y axes need to be reversed.
    machine.config_ReverseDirection(AXIS_X);
    machine.config_ReverseDirection(AXIS_Y);

    // PPU (pulses per unit) is a virtual length unit defined by your application.
    // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
    machine.config_PPU(AXIS_X, 80.0);
    machine.config_PPU(AXIS_Y, 80.0);
    machine.config_PPU(AXIS_Z, 1600.0);

    // The machine must be turned on before it can be controlled.
    machine.machineOn();
}
```

```

// Use Enc3 as the MPG source.
machine.beginMpg(Enc3);

// Select the axis to be driven by the MPG hand wheel.
machine.setMpgAxis(AXIS_X);
// machine.setMpgAxis(AXIS_Y);
// machine.setMpgAxis(AXIS_Z);

// Feedrate in MPG mode.
machine.setMpgSpeed(600);

// MPG ratio is the ratio between hand-wheel counts and machine travel; it must be greater
than 0.
machine.setMpgRatio(1);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Mpg position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}

```

getJogPos()

Description

Returns the target position of the machine in JOG / MPG mode, either for one axis or for all three axes through reference arguments.

Syntax

```
machine.getJogPos(axis);
machine.getJogPos(x, y, z);
```

Parameters

- `machine`: a `Machine` object.
- `axis`: the axis to configure: `AXIS_X`, `AXIS_Y` or `AXIS_Z`.
- `x`: reference to a `double` that receives the X position.
- `y`: reference to a `double` that receives the Y position.
- `z`: reference to a `double` that receives the Z position.

Returns

`double`: position of the requested axis (single-axis form). The three-axis form writes the positions into `x`, `y`, `z`.

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  // In this example the X and Y axes need to be reversed.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);
```

```

// The machine must be turned on before it can be controlled.
machine.machineOn();

// Put the machine in Jog mode, triggered by the buttons wired to these pins.
machine.beginJog(27, 28, 32, 3, 16, 17, false);

// If incJog mode is used, set the jog offset to define how many millimeters to move per
click.
// machine.beginJog(27, 28, 32, 3, 16, 17, true);
// machine.setJogOffset(1.0);

// Feedrate in Jog mode.
machine.setJogSpeed(600);

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Jog position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}

```

2.1.6 Feedrate Override

Change the feedrate of queued motion in real time.

Functions:

- [feedrateOverride\(\)](#)
- [getFeedrateOverride\(\)](#)

feedrateOverride()

Description

Rescales the feedrate of all motion blocks in the buffer that have not been executed yet, and recalculates the trajectory. Use it to slow down or speed up a running program in real time.

Syntax

```
machine.feedrateOverride(ratio);
```

Parameters

- **machine**: a **Machine** object.
- **ratio**: feedrate override ratio; must be greater than 0 (**1.0** = 100 %).

Returns

None.

Example

```
#include "Motion86.h"
#include "Encoder.h"
#include "SD.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

// The G-code file stored on the SD card.
char *filename = "auto.gcode";
File gfile;
char buf[256];
int ptr;
bool working;

// Set the initial feedrate override ratio.
volatile double FeedrateRatio = 1.0;

bool enc_triggered = false;

void encoder_callback(int flag);

void setup() {
  while (!Serial);
  SD.begin();
```

```

// In this example the hand wheel is connected to Enc3.
Enc3.begin(MODE_AB_PHASE);
Enc3.setRange(1); // trigger an interrupt on every count
Enc3.attachInterrupt(encoder_callback);

pinMode(EnablePin, OUTPUT);

// Reverse the direction of an axis if required.
// In this example the X and Y axes need to be reversed.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// PPU (pulses per unit) is a virtual length unit defined by your application.
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// Set the software limits of the machine.
machine.config_PosLimit(AXIS_X, 0, 300);
machine.config_PosLimit(AXIS_Y, 0, 200);
machine.config_PosLimit(AXIS_Z, 0, 300);

// Set the limit-switch pins that define the home point.
machine.config_HomePins(2, 7, 8);

// The machine must be turned on before it can be controlled.
machine.machineOn();

// Enable the software limits.
machine.enableSoftLimit();

// Set the homing feedrate.
machine.setHomeSpeed(1000, 1000, 200);

// Open the G-code file on the SD card.
if (SD.exists(filename)) {
    gfile = SD.open(filename);
    working = true;
} else {
    Serial.print("File not exists: ");
    Serial.println(filename);
}

```

```

    while (1);
}

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);

// Return to the home point defined by the limit switches.
machine.home();
}

void loop() {
    // Read and parse the G-code file.
    if (working && !machine.isCmdBufferFull()) {
        ptr = 0;
        while (gfile.available()) {
            buf[ptr] = gfile.read();
            if (buf[ptr] == '\n') {
                buf[ptr + 1] = '\0';
                machine.gcode(buf);
                break;
            } else {
                ptr ++;
            }
        }
        if (!gfile.available())
        {
            Serial.println("GCODE FINISH");
            gfile.close();
            working = false;
        }
    }

    // Override the feedrate when the encoder interrupt fires.
    if (enc_triggered) {
        if (FeedrateRatio > 2.0) FeedrateRatio = 2.0;
        if (FeedrateRatio < 0.1) FeedrateRatio = 0.1;
        machine.feedrateOverride(FeedrateRatio);
        enc_triggered = false;
        Serial.print("FeedrateRatio = ");
        Serial.println(machine.getFeedrateOverride());
    }
}

```

```
void encoder_callback(int flag) {  
    // The feedrate ratio must be greater than 0.  
    if (flag == INTR_OVERFLOW && FeedrateRatio < 2.0)  
        FeedrateRatio += 0.01;  
    else if (flag == INTR_UNDERFLOW && FeedrateRatio > 0.0)  
        FeedrateRatio -= 0.01;  
    enc_triggered = true;  
}
```

getFeedrateOverride()

Description

Returns the current feedrate override ratio.

Syntax

```
machine.getFeedrateOverride();
```

Parameters

- `machine`: a `Machine` object.

Returns

`double`: feedrate override ratio (1.0 = 100 %).

Example

```
#include "Motion86.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulses per unit) is a virtual length unit defined by your application.
  // Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // The machine must be turned on before it can be controlled.
  machine.machineOn();

  // Enable the stepper drivers.
  digitalWrite(EnablePin, LOW);
}

void loop() {
```

```
// Run the next moves at half speed, then report the ratio.  
machine.feedrateOverride(0.5);  
machine.line(10, 10, 0, 400);  
Serial.print("Feedrate override = ");  
Serial.println(machine.getFeedrateOverride());  
while (machine.isMoving());  
machine.feedrateOverride(1.0);  
machine.line(0, 0, 0, 400);  
while (machine.isMoving());  
}
```

2.1.7 Axis Following (Electronic Gear)

Make a second machine follow this one with programmable gear ratios.

Functions:

- [config_EGearSlave\(\)](#)

config_EGearSlave()

Description

Registers another `Machine` as the electronic-gear slave of this machine. When enabled, the target machine follows each axis of this machine according to the given gear ratios, so a single command stream drives both machines. Configure it before `machineOn()`.

Syntax

```
machine.config_EGearSlave(target, xRatio, yRatio, zRatio);
```

Parameters

- `machine`: a `Machine` object.
- `target`: the `Machine` object that will follow this machine.
- `xRatio`: electronic gear ratio of the X axis.
- `yRatio`: electronic gear ratio of the Y axis.
- `zRatio`: electronic gear ratio of the Z axis.

Returns

`bool`: `true` if the configuration succeeded; `false` if the machine does not exist or the machine has already been turned on.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"
#include "SD.h"

// Create the Machine object. Up to three machines (0~2) with three axes each are available.
// This example uses machine 0 and machine 1.
Machine machine(0);
Machine slave_machine(1);

// Stepper driver enable pin.
int EnablePin = 12;

// The G-code file stored on the SD card.
char *filename = "auto.gcode";
File gfile;
char buf[256];
int ptr;
bool working;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);
```

```

// Reverse the direction of an axis if required.
// In this example the X and Y axes need to be reversed.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// PPU (pulses per unit) is a virtual length unit defined by your application.
// Here one unit on the X axis is 80 pulses, i.e. 1 mm in the real application.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// Set the software limits of the machine.
machine.config_PosLimit(AXIS_X, 0, 300);
machine.config_PosLimit(AXIS_Y, 0, 200);
machine.config_PosLimit(AXIS_Z, 0, 300);

// Set the limit-switch pins that define the home point.
machine.config_HomePins(2, 7, 8);

// In this example slave_machine needs its Y axis reversed.
slave_machine.config_ReverseDirection(AXIS_Y);

// Set the PPU (pulses per unit) of slave_machine.
slave_machine.config_PPU(AXIS_X, 80.0);
slave_machine.config_PPU(AXIS_Y, 80.0);
slave_machine.config_PPU(AXIS_Z, 3200.0);

// Set the software limits of slave_machine.
slave_machine.config_PosLimit(AXIS_X, 0, 200);
slave_machine.config_PosLimit(AXIS_Y, 0, 200);
slave_machine.config_PosLimit(AXIS_Z, 0, 300);

// Set the limit-switch pins that define the home point of slave_machine.
slave_machine.config_HomePins(21, 22, 23);

// Register slave_machine as the electronic-gear slave of machine: it will follow machine's
motion by the given ratios.
machine.config_EGearSlave(slave_machine, 1.5, 0.5, 1.0);

// The machine must be turned on before it can be controlled.
machine.machineOn();

```

```

// Enable the software limits.
machine.enableSoftLimit();

// Set the homing feedrate.
machine.setHomeSpeed(1000, 1000, 200);
slave_machine.setHomeSpeed(1000, 1000, 100);

// Open the G-code file on the SD card.
if (SD.exists(filename)) {
    gfile = SD.open(filename);
    working = true;
} else {
    Serial.print("File not exists: ");
    Serial.println(filename);
    while (1);
}

// Enable the stepper drivers.
digitalWrite(EnablePin, LOW);

// Return to the home point defined by the limit switches.
machine.home();
}

void loop() {
    // Read and parse the G-code file.
    // slave_machine reproduces the G-code path scaled x1.5 wide and x0.5 high.
    if (working && !machine.isCmdBufferFull()) {
        ptr = 0;
        while (gfile.available()) {
            buf[ptr] = gfile.read();
            if (buf[ptr] == '\n') {
                buf[ptr + 1] = '\0';
                machine.gcode(buf);
                break;
            } else {
                ptr ++;
            }
        }
        if (!gfile.available())
        {
            Serial.println("GCODE FINISH");
        }
    }
}

```

```
gfile.close();  
working = false;  
}  
}  
}
```

2.1.8 Calculation Tools

Helpers to derive the PPU from mechanical parameters.

Functions:

- [computePPU_Belt\(\)](#)
- [computePPU_LeadScrew\(\)](#)

computePPU_Belt()

Description

Computes the PPU (pulses per mm) of a belt-driven axis from the motor pulses per revolution, the belt pitch and the pulley tooth count. The result can be passed directly to `config_PPU()`.

Syntax

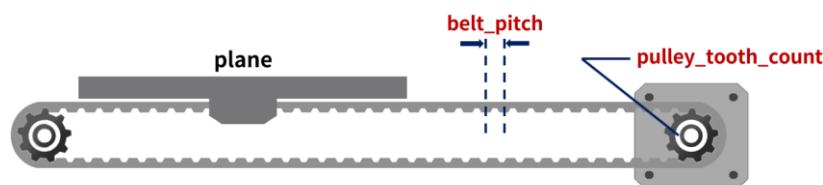
```
machine.computePPU_Belt(pulses_per_revolution, belt_pitch, pulley_tooth_count);
```

Parameters

- `machine`: a `Machine` object.
- `pulses_per_revolution`: number of pulses the 86Duino must output for one revolution of the stepper motor.
- `belt_pitch`: belt pitch in mm.
- `pulley_tooth_count`: number of teeth of the pulley that rotates with the stepper motor.

Returns

`double`: the computed PPU (pulses per mm).



Example

```
#include "Motion86.h"

Machine machine(0);

void setup() {
  // 200-step motor with 16x micro-stepping = 3200 pulses/rev,
  // GT2 belt (2 mm pitch) on a 20-tooth pulley.
  double ppu = machine.computePPU_Belt(3200, 2.0, 20);  // = 80 pulses/mm
  Serial.print("PPU = ");
  Serial.println(ppu);

  machine.config_PPU(AXIS_X, ppu);
  machine.config_PPU(AXIS_Y, ppu);
  machine.config_PPU(AXIS_Z, 1600.0);
  machine.machineOn();
}

void loop() {
}
```

computePPU_LeadScrew()

Description

Computes the PPU (pulses per mm) of a lead-screw axis from the motor pulses per revolution, the screw pitch and the gearbox ratio. The result can be passed directly to `config_PPU()`.

Syntax

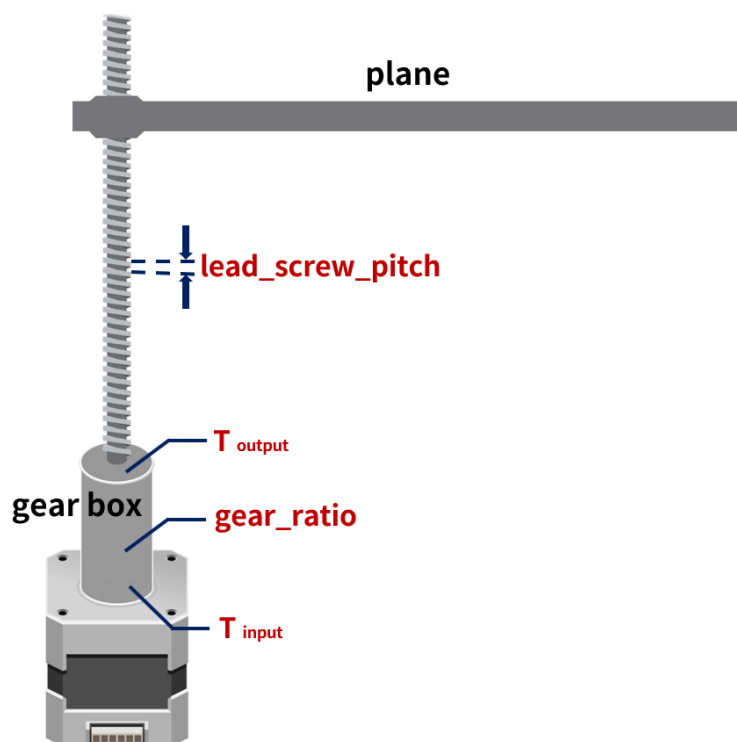
```
machine.computePPU_LeadScrew(pulses_per_revolution, leadscrew_pitch, gear_ratio);
```

Parameters

- `machine`: a `Machine` object.
- `pulses_per_revolution`: number of pulses the 86Duino must output for one revolution of the stepper motor.
- `leadscrew_pitch`: pitch of the lead screw in mm.
- `gear_ratio`: gearbox ratio = output gear teeth / input gear teeth ($T_{\text{output}} / T_{\text{input}}$); use 1 for a direct drive.

Returns

`double`: the computed PPU (pulses per mm).



Example

```
#include "Motion86.h"

Machine machine(0);

void setup() {
  while (!Serial);
```

```
// 3200 pulses/rev, 2 mm pitch lead screw, direct drive.  
double ppu = machine.computePPU_LeadScrew(3200, 2.0, 1.0); // = 1600 pulses/mm  
Serial.print("PPU = ");  
Serial.println(ppu);  
  
machine.config_PPU(Axis_X, 80.0);  
machine.config_PPU(Axis_Y, 80.0);  
machine.config_PPU(Axis_Z, ppu);  
machine.machineOn();  
}  
  
void loop() {  
}
```

2.2 EcatMachine Class

`EcatMachine` is a plugin that inherits from `Machine` and implements a dedicated EtherCAT motion controller on top of 86EVA. It lets you control EtherCAT servo drives with the Motion86 API without any knowledge of the EtherCAT protocol: the drives are bound to the axes with `attach_EcatServo_*`(), and `machineOn()` performs the whole EtherCAT initialisation and switches the drives to CiA 402 CSP mode.

Requirements:

- An 86EVA project that generates `myeva.h` for your slave topology (86Duino Coding IDE 501+).
- Include order: `myeva.h` **must** be included before `Motion86_Ethercat.h`, so that the `EVA`, `EcatMaster` and virtual device objects exist when the plugin is compiled.
- Only the custom axis groups `CUSTOM_MACHINE_0~CUSTOM_MACHINE_2` are accepted.
- All `attach_*` methods must be called before `machineOn()`.

```
#include "myeva.h"           // must come first
#include "Motion86_Ethercat.h"

EcatMachine machine(CUSTOM_MACHINE_0);
```

Pin numbers passed to the `attach_EcatHomePin_*`() and `attach_EcatJogPins_*`() methods refer to the **Virtual Arduino** numbering defined in 86EVA, not to physical 86Duino pins. After attaching jog pins or an MPG device, use the argument-less `beginJog()` / `beginMpg()` forms.

Groups:

- [2.2.1 Constructor / Destructor](#)
- [2.2.2 Attach EcatServo Method](#)
- [2.2.3 Attach Home Pin Method](#)
- [2.2.4 Attach Jog Pins Method](#)
- [2.2.5 Attach Mpg Method](#)
- [2.2.6 Basic Method \(overridden\)](#)

2.2.1 Constructor / Destructor

Functions:

- [EcatMachine\(\)](#)
- [~EcatMachine\(\)](#)

EcatMachine()

Description

Constructs an EtherCAT motion controller on a custom axis group. `EcatMachine` inherits from `Machine`, so every method of Chapter 2.1 is available. It requires an 86EVA project and the include order shown below (`myeva.h` first).

Only `CUSTOM_MACHINE_0`, `CUSTOM_MACHINE_1` and `CUSTOM_MACHINE_2` are accepted.

Syntax

```
#include "myeva.h"
#include "Motion86_Ethercat.h"
EcatMachine machine(axis_group);
```

Parameters

- `axis_group`: `CUSTOM_MACHINE_0`, `CUSTOM_MACHINE_1` or `CUSTOM_MACHINE_2`.

Returns

None. Use `isValid()` to check whether the object was created successfully.

Example

```
#include "myeva.h"
#include "Motion86_Ethercat.h"

// EcatMachine only accepts CUSTOM_MACHINE_0 ~ CUSTOM_MACHINE_2.
EcatMachine machine(CUSTOM_MACHINE_0);

void setup() {
  // Start the EtherCAT master and the 86EVA virtual devices.
  EVA.begin();

  // Bind the virtual servos to the axes (before machineOn()).
  machine.attach_EcatServo_X(VirtualServo1);
  machine.attach_EcatServo_Y(VirtualServo2);
  machine.attach_EcatServo_Z(VirtualServo3);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);

  // PPU: encoder counts per mm.
  machine.config_PPU(AXIS_X, 1600.0);
  machine.config_PPU(AXIS_Y, 1600.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // machineOn() initialises the EtherCAT drives and switches them to CSP mode.
  machine.machineOn();
```

```
machine.setDefaultFeedrate(400);  
}  
  
void loop() {  
  // Move left to (-10, 0).  
  machine.line(-10, 0, 0);  
  // Clockwise 90-degree arc with radius 10 to (0, 10).  
  machine.arcXY(10, 0, 10, true);  
  // Move down to (0, -10).  
  machine.line(0, -10, 0);  
  // Counter-clockwise 90-degree arc with radius 10 to (10, 0).  
  machine.arcXY(10, 10, 0, false);  
  // Move back to (0, 0).  
  machine.line(0, 0, 0);  
  // Wait until the planned motion is finished.  
  while (machine.isMoving());  
}
```

~EcatMachine()

Description

Destroys the `EcatMachine` object. In this plugin the destructor does not need to release any resources; the base class destructor releases the axis group.

Syntax

```
delete machine;
```

Parameters

- `machine`: a pointer to an `EcatMachine` object created with `new`.

Returns

None.

See also: [~Machine\(\)](#)

2.2.2 Attach EcatServo Method

Bind 86EVA virtual servos or CiA 402 devices to the X, Y and Z axes.

Functions:

- [attach_EcatServo_X\(\)](#)
- [attach_EcatServo_Y\(\)](#)
- [attach_EcatServo_Z\(\)](#)

attach_EcatServo_X()

Description

Binds a servo drive to the X axis. The servo can be an 86EVA virtual servo (`MyVirtualServo`, e.g. `VirtualServo1`) or a raw CiA 402 EtherCAT device (`EthercatDevice_CiA402`), passed by reference or by pointer.

Attaching a virtual servo replaces any raw device previously attached to the axis, and vice versa. An axis without an attached servo is not driven by the plugin.

Syntax

```
machine.attach_EcatServo_X(vServo);
machine.attach_EcatServo_X(&vServo);
machine.attach_EcatServo_X(ecatDev);
machine.attach_EcatServo_X(&ecatDev);
```

Parameters

- `machine`: an `EcatMachine` object.
- `vServo`: a `MyVirtualServo` object defined in the 86EVA configuration.
- `ecatDev`: an `EthercatDevice_CiA402` object.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

Example

```
#include "myeva.h"
#include "Motion86_Ethercat.h"

// EcatMachine only accepts CUSTOM_MACHINE_0 ~ CUSTOM_MACHINE_2.
EcatMachine machine(CUSTOM_MACHINE_0);

void setup() {
  // Start the EtherCAT master and the 86EVA virtual devices.
  EVA.begin();

  // Bind the virtual servos to the axes (before machineOn()).
  machine.attach_EcatServo_X(VirtualServo1);
  machine.attach_EcatServo_Y(VirtualServo2);
  machine.attach_EcatServo_Z(VirtualServo3);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);

  // PPU: encoder counts per mm.
  machine.config_PPU(AXIS_X, 1600.0);
```

```
machine.config_PPU(AXIS_Y, 1600.0);
machine.config_PPU(AXIS_Z, 1600.0);

// machineOn() initialises the EtherCAT drives and switches them to CSP mode.
machine.machineOn();
machine.setDefaultFeedrate(400);
}

void loop() {
  // Move left to (-10, 0).
  machine.line(-10, 0, 0);
  // Clockwise 90-degree arc with radius 10 to (0, 10).
  machine.arcXY(10, 0, 10, true);
  // Move down to (0, -10).
  machine.line(0, -10, 0);
  // Counter-clockwise 90-degree arc with radius 10 to (10, 0).
  machine.arcXY(10, 10, 0, false);
  // Move back to (0, 0).
  machine.line(0, 0, 0);
  // Wait until the planned motion is finished.
  while (machine.isMoving());
}
```

attach_EcatServo_Y()

Description

Binds a servo drive to the Y axis. The servo can be an 86EVA virtual servo (`MyVirtualServo`, e.g. `VirtualServo1`) or a raw CiA 402 EtherCAT device (`EthercatDevice_CiA402`), passed by reference or by pointer.

Attaching a virtual servo replaces any raw device previously attached to the axis, and vice versa. An axis without an attached servo is not driven by the plugin.

Syntax

```
machine.attach_EcatServo_Y(vServo);  
machine.attach_EcatServo_Y(&vServo);  
machine.attach_EcatServo_Y(ecatDev);  
machine.attach_EcatServo_Y(&ecatDev);
```

Parameters

- `machine`: an `EcatMachine` object.
- `vServo`: a `MyVirtualServo` object defined in the 86EVA configuration.
- `ecatDev`: an `EthercatDevice_CiA402` object.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

attach_EcatServo_Z()

Description

Binds a servo drive to the Z axis. The servo can be an 86EVA virtual servo (`MyVirtualServo`, e.g. `VirtualServo1`) or a raw CiA 402 EtherCAT device (`EthercatDevice_CiA402`), passed by reference or by pointer.

Attaching a virtual servo replaces any raw device previously attached to the axis, and vice versa. An axis without an attached servo is not driven by the plugin.

Syntax

```
machine.attach_EcatServo_Z(vServo);  
machine.attach_EcatServo_Z(&vServo);  
machine.attach_EcatServo_Z(ecatDev);  
machine.attach_EcatServo_Z(&ecatDev);
```

Parameters

- `machine`: an `EcatMachine` object.
- `vServo`: a `MyVirtualServo` object defined in the 86EVA configuration.
- `ecatDev`: an `EthercatDevice_CiA402` object.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

2.2.3 Attach Home Pin Method

Declare the Virtual Arduino inputs used as home switches.

Functions:

- [attach_EcatHomePin_X\(\)](#)
- [attach_EcatHomePin_Y\(\)](#)
- [attach_EcatHomePin_Z\(\)](#)

attach_EcatHomePin_X()

Description

Assigns a Virtual Arduino digital input as the home switch of the X axis. `home()` reads it through `EVA.digitalRead()`. Attaching at least one home pin authorises the homing procedure on this machine.

Syntax

```
machine.attach_EcatHomePin_X(limit);
```

Parameters

- `machine`: an `EcatMachine` object.
- `limit`: Virtual Arduino pin number (≥ 0) defined in the 86EVA configuration.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

Returns `false` if `limit` is negative.

Example

```
#include "myeva.h"
#include "Motion86_Ethercat.h"

// EcatMachine only accepts CUSTOM_MACHINE_0 ~ CUSTOM_MACHINE_2.
EcatMachine machine(CUSTOM_MACHINE_0);

void setup() {
  // Start the EtherCAT master and the 86EVA virtual devices.
  EVA.begin();

  // Bind the virtual servos to the axes (before machineOn()).
  machine.attach_EcatServo_X(VirtualServo1);
  machine.attach_EcatServo_Y(VirtualServo2);
  machine.attach_EcatServo_Z(VirtualServo3);

  // Home switches wired to Virtual Arduino inputs 0, 1, 2.
  machine.attach_EcatHomePin_X(0);
  machine.attach_EcatHomePin_Y(1);
  machine.attach_EcatHomePin_Z(2);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);

  // PPU: encoder counts per mm.
  machine.config_PPU(AXIS_X, 1600.0);
```

```
machine.config_PPU(AXIS_Y, 1600.0);
machine.config_PPU(AXIS_Z, 1600.0);

// machineOn() initialises the EtherCAT drives and switches them to CSP mode.
machine.machineOn();
machine.setDefaultFeedrate(400);

machine.setHomeSpeed(20, 20, 10);
machine.home();
}

void loop() {
    // Motion control code...
}
```

attach_EcatHomePin_Y()

Description

Assigns a Virtual Arduino digital input as the home switch of the Y axis. `home()` reads it through `EVA.digitalRead()`. Attaching at least one home pin authorises the homing procedure on this machine.

Syntax

```
machine.attach_EcatHomePin_Y(limit);
```

Parameters

- `machine`: an `EcatMachine` object.
- `limit`: Virtual Arduino pin number (≥ 0) defined in the 86EVA configuration.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

Returns `false` if `limit` is negative.

attach_EcatHomePin_Z()

Description

Assigns a Virtual Arduino digital input as the home switch of the Z axis. `home()` reads it through `EVA.digitalRead()`. Attaching at least one home pin authorises the homing procedure on this machine.

Syntax

```
machine.attach_EcatHomePin_Z(limit);
```

Parameters

- `machine`: an `EcatMachine` object.
- `limit`: Virtual Arduino pin number (≥ 0) defined in the 86EVA configuration.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

Returns `false` if `limit` is negative.

2.2.4 Attach Jog Pins Method

Declare the Virtual Arduino inputs used as jog switches.

Functions:

- [attach_EcatJogPins_X\(\)](#)
- [attach_EcatJogPins_Y\(\)](#)
- [attach_EcatJogPins_Z\(\)](#)

attach_EcatJogPins_X()

Description

Assigns two Virtual Arduino digital inputs as the positive and negative jog switches of the X axis, each with its own active level. Afterwards enter JOG mode with `beginJog()` / `beginJog(incJog)` (the form without pins).

Syntax

```
machine.attach_EcatJogPins_X(positivePin, negativePin);
machine.attach_EcatJogPins_X(positivePin, negativePin, positivePolarity,
negativePolarity);
```

Parameters

- `machine`: an `EcatMachine` object.
- `positivePin`: Virtual Arduino pin of the X+ switch.
- `negativePin`: Virtual Arduino pin of the X- switch.
- `positivePolarity`: `ACTIVE_HIGH` (default) or `ACTIVE_LOW`.
- `negativePolarity`: `ACTIVE_HIGH` (default) or `ACTIVE_LOW`.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

Example

```
#include "myeva.h"
#include "Motion86_Ethercat.h"

// EcatMachine only accepts CUSTOM_MACHINE_0 ~ CUSTOM_MACHINE_2.
EcatMachine machine(CUSTOM_MACHINE_0);

void setup() {
  // Start the EtherCAT master and the 86EVA virtual devices.
  EVA.begin();

  // Bind the virtual servos to the axes (before machineOn()).
  machine.attach_EcatServo_X(VirtualServo1);
  machine.attach_EcatServo_Y(VirtualServo2);
  machine.attach_EcatServo_Z(VirtualServo3);

  // Jog buttons on Virtual Arduino inputs 4~9, active low.
  machine.attach_EcatJogPins_X(4, 5, ACTIVE_LOW, ACTIVE_LOW);
  machine.attach_EcatJogPins_Y(6, 7, ACTIVE_LOW, ACTIVE_LOW);
  machine.attach_EcatJogPins_Z(8, 9, ACTIVE_LOW, ACTIVE_LOW);

  // Reverse the direction of an axis if required.
```

```
machine.config_ReverseDirection(AXIS_X);

// PPU: encoder counts per mm.
machine.config_PPU(AXIS_X, 1600.0);
machine.config_PPU(AXIS_Y, 1600.0);
machine.config_PPU(AXIS_Z, 1600.0);

// machineOn() initialises the EtherCAT drives and switches them to CSP mode.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Continuous jog mode using the attached pins.
machine.beginJog();
machine.setJogSpeed(100);
}

void loop() {
  Serial.print("Jog position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}
```

attach_EcatJogPins_Y()

Description

Assigns two Virtual Arduino digital inputs as the positive and negative jog switches of the Y axis, each with its own active level. Afterwards enter JOG mode with `beginJog()` / `beginJog(incJog)` (the form without pins).

Syntax

```
machine.attach_EcatJogPins_Y(positivePin, negativePin);  
machine.attach_EcatJogPins_Y(positivePin, negativePin, positivePolarity,  
negativePolarity);
```

Parameters

- `machine`: an `EcatMachine` object.
- `positivePin`: Virtual Arduino pin of the Y+ switch.
- `negativePin`: Virtual Arduino pin of the Y- switch.
- `positivePolarity`: `ACTIVE_HIGH` (default) or `ACTIVE_LOW`.
- `negativePolarity`: `ACTIVE_HIGH` (default) or `ACTIVE_LOW`.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

attach_EcatJogPins_Z()

Description

Assigns two Virtual Arduino digital inputs as the positive and negative jog switches of the Z axis, each with its own active level. Afterwards enter JOG mode with `beginJog()` / `beginJog(incJog)` (the form without pins).

Syntax

```
machine.attach_EcatJogPins_Z(positivePin, negativePin);  
machine.attach_EcatJogPins_Z(positivePin, negativePin, positivePolarity,  
negativePolarity);
```

Parameters

- `machine`: an `EcatMachine` object.
- `positivePin`: Virtual Arduino pin of the Z+ switch.
- `negativePin`: Virtual Arduino pin of the Z- switch.
- `positivePolarity`: `ACTIVE_HIGH` (default) or `ACTIVE_LOW`.
- `negativePolarity`: `ACTIVE_HIGH` (default) or `ACTIVE_LOW`.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

2.2.5 Attach Mpg Method

Declare the hand-wheel source.

Functions:

- [attach_EcatMpg\(\)](#)

attach_EcatMpg()

Description

Attaches the hand-wheel source used in MPG mode: an 86EVA virtual MPG (`MyVirtualMpg`) or a virtual encoder (`MyVirtualEncoder`), by reference or by pointer. Only one of the two can be attached at a time – attaching one clears the other. Afterwards enter MPG mode with `beginMpg()` (the form without argument).

Syntax

```
machine.attach_EcatMpg(vMpg);
machine.attach_EcatMpg(&vMpg);
machine.attach_EcatMpg(vEnc);
machine.attach_EcatMpg(&vEnc);
```

Parameters

- `machine`: an `EcatMachine` object.
- `vMpg`: a `MyVirtualMpg` object; its counter range is treated as 32-bit.
- `vEnc`: a `MyVirtualEncoder` object; its range is taken from `getRange()`.

Returns

`bool`: `true` on success; `false` if the machine does not exist or has already been turned on.

All `attach_*` methods must be called **before** `machineOn()`.

Example

```
#include "myeva.h"
#include "Motion86_Ethercat.h"

// EcatMachine only accepts CUSTOM_MACHINE_0 ~ CUSTOM_MACHINE_2.
EcatMachine machine(CUSTOM_MACHINE_0);

void setup() {
  // Start the EtherCAT master and the 86EVA virtual devices.
  EVA.begin();

  // Bind the virtual servos to the axes (before machineOn()).
  machine.attach_EcatServo_X(VirtualServo1);
  machine.attach_EcatServo_Y(VirtualServo2);
  machine.attach_EcatServo_Z(VirtualServo3);

  // Hand wheel on the EtherCAT HID slave.
  machine.attach_EcatMpg(VirtualMpg1);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);
```

```
// PPU: encoder counts per mm.
machine.config_PPU(AXIS_X, 1600.0);
machine.config_PPU(AXIS_Y, 1600.0);
machine.config_PPU(AXIS_Z, 1600.0);

// machineOn() initialises the EtherCAT drives and switches them to CSP mode.
machine.machineOn();
machine.setDefaultFeedrate(400);

machine.beginMpg();
machine.setMpgAxis(AXIS_X);
machine.setMpgSpeed(100);
machine.setMpgRatio(0.01);
}

void loop() {
  Serial.print("MPG position = ");
  Serial.println(machine.getJogPos(AXIS_X));
  delay(500);
}
```

2.2.6 Basic Method (overridden)

EtherCAT-specific start-up and shutdown.

Functions:

- [machineOn\(\) \[EcatMachine\]](#)
- [machineOff\(\) \[EcatMachine\]](#)

machineOn() [EcatMachine]

Description

Overrides `Machine::machineOn()` and performs the complete EtherCAT start-up:

1. Reads the cycle time of the EtherCAT master and applies it with `config_CustomActuators_CycleTime()`.
2. Stops the master, resolves each attached virtual servo to its CiA 402 device and sets the DC cycle of every servo.
3. Installs the cyclic callback that calls `updateAxes()` for every `EcatMachine` and then chains to the 86EVA cyclic callback.
4. Restarts the master, switches every attached servo to CiA 402 CSP (Cyclic Synchronous Position) mode and enables it.
5. Enters MachineOn mode. Configuration and `attach_*` calls are locked afterwards.

Syntax

```
machine.machineOn();
```

Parameters

- `machine`: an `EcatMachine` object.

Returns

None.

machineOff() [EcatMachine]

Description

Overrides `Machine::machineOff()`: leaves MachineOn mode and then disables every attached servo.

Syntax

```
machine.machineOff();
```

Parameters

- `machine`: an `EcatMachine` object.

Returns

None.

2.3 Custom Machine Tools

A Custom Machine connects Motion86 to actuators that are reached over a communication bus instead of local pulse outputs.

Create the `Machine` on `CUSTOM_MACHINE_0~CUSTOM_MACHINE_2` and provide the actuator and I/O access in one of two ways:

- **Callbacks** (2.3.1) – register plain C functions with the `config*_Callback()` methods and call `updateAxes()` yourself every cycle.
- **Plugin** (2.3.2) – derive a class from `Machine` and override the virtual hook methods. `EcatMachine` is implemented this way.

In both cases `updateAxes()` must be called at exactly the cycle time set with `config_CustomActuators_CycleTime()`, normally from the real-time cyclic interrupt of the fieldbus master. Cycle jitter directly degrades path accuracy.

Groups:

- [2.3.1 Callback Functions](#)
- [2.3.2 Plugin Override Points](#)

2.3.1 Callback Functions

These `Machine` methods are only accepted on a Custom Machine and must be called before `machineOn()`.

Functions:

- [config_CustomActuators_Callbacks\(\)](#)
- [config_CustomActuators_CycleTime\(\)](#)
- [config_HomeController_Callback\(\)](#)
- [config_JogController_Callback\(\)](#)
- [config_MpgController_Callback\(\)](#)
- [updateAxes\(\)](#)

config_CustomActuators_Callbacks()

Description

Registers the position write / read callbacks of a Custom Machine. On every cycle `updateAxes()` calls `setCB` with the new target position of each axis and `getCB` to read back the actual position. Custom actuators must be used together with `updateAxes()`, which has to be called at a fixed cycle time to keep the machine stable.

Syntax

```
machine.config_CustomActuators_Callbacks(setCB, getCB);
```

Parameters

- `machine`: a `Machine` object.
- `setCB`: `void setCB(int axis, long pos)` – `axis` is 0 / 1 / 2 for X / Y / Z, `pos` is the target position in pulses (counts).
- `getCB`: `long getCB(int axis)` – returns the actual position of the axis in pulses (counts).

Returns

`bool`: `true` on success; `false` if the machine does not exist, has already been turned on, or is not a Custom Machine.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
#include "Motion86.h"
#include "Ethercat.h"

// A Machine on a custom axis group, driven through callbacks.
Machine machine(CUSTOM_MACHINE_0);
EthercatMaster master;
EthercatDevice_CiA402 servo[3];

// Position callbacks: axis is 0/1/2 for X/Y/Z, positions are in pulses / counts.
void setRemoteTargetPos(int axis, long pos) { servo[axis].setTargetPosition(pos); }
long getRemoteTargetPos(int axis)           { return servo[axis].getPositionActualValue(); }

// Home switch callback: digital input 0 of each drive.
int myHomeCommand(homeCommand_t feat) {
    if (feat == READ_HOME_X) return servo[0].getDigitalInputs() & 0x01;
    if (feat == READ_HOME_Y) return servo[1].getDigitalInputs() & 0x01;
    if (feat == READ_HOME_Z) return servo[2].getDigitalInputs() & 0x01;
    return 0;
}

// Called by the EtherCAT master every cycle (500 us).
void MyCyclicCallback() {
```

```

machine.updateAxes();
}

void setup() {
  master.begin();
  for (int i = 0; i < 3; i++) {
    servo[i].attach(0, i, master);
    servo[i].setDc(500000);
  }

  machine.config_ReverseDirection(Axis_X);
  machine.config_PPU(Axis_X, 1600.0);
  machine.config_PPU(Axis_Y, 1600.0);
  machine.config_PPU(Axis_Z, 1600.0);

  // Custom actuator hooks: cycle time must match the master cycle.
  machine.config_CustomActuators_CycleTime(500000);
  machine.config_CustomActuators_Callbacks(setRemoteTargetPos, getRemoteTargetPos);
  machine.config_HomeController_Callback(myHomeCommand);

  master.attachRTCCyclicCallback(MyCyclicCallback, true);
  master.start(500000, ECAT_SYNC);
  for (int i = 0; i < 3; i++) {
    servo[i].enable();
    servo[i].setCiA402Mode(CIA402_CSP_MODE);
  }

  // The machine must be turned on before it can be controlled.
  machine.machineOn();
  machine.setDefaultFeedrate(400);
}

void loop() {
  // Move left to (-10, 0).
  machine.line(-10, 0, 0);
  // Clockwise 90-degree arc with radius 10 to (0, 10).
  machine.arcXY(10, 0, 10, true);
  // Move down to (0, -10).
  machine.line(0, -10, 0);
  // Counter-clockwise 90-degree arc with radius 10 to (10, 0).
  machine.arcXY(10, 10, 0, false);
  // Move back to (0, 0).

```

```
machine.line(0, 0, 0);  
// Wait until the planned motion is finished.  
while (machine.isMoving());  
}
```

config_CustomActuators_CycleTime()

Description

Declares the period at which `updateAxes()` will be called for a Custom Machine. Motion86 uses this period for interpolation, so it must match the real call period exactly – normally the cycle time of the fieldbus master.

`EcatMachine::machineOn()` sets this automatically from the EtherCAT master.

Syntax

```
machine.config_CustomActuators_CycleTime(period);
```

Parameters

- `machine`: a `Machine` object.
- `period`: cycle period; pass the same value as the cycle time given to the EtherCAT master (e.g. `500000` for a 500 μ s cycle).

Returns

`bool`: `true` on success; `false` if the machine does not exist, has already been turned on, or is not a Custom Machine.

Make sure the machine exists and configure it **before** calling `machineOn()`.

config_HomeController_Callback()

Description

Registers the callback that `home()` uses to read the home switches of a Custom Machine.

Syntax

```
machine.config_HomeController_Callback(cb);
```

Parameters

- `machine`: a `Machine` object.
- `cb: int cb(homeCommand_t cmd)` – returns the switch state for `READ_HOME_X`, `READ_HOME_Y` or `READ_HOME_Z`. Pass `NULL` (default) to remove the callback.

Returns

`bool`: `true` on success; `false` if the machine does not exist, has already been turned on, or is not a Custom Machine.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
int myHomeCommand(homeCommand_t feat) {
    if (feat == READ_HOME_X) return servo[0].getDigitalInputs() & 0x01;
    if (feat == READ_HOME_Y) return servo[1].getDigitalInputs() & 0x01;
    if (feat == READ_HOME_Z) return servo[2].getDigitalInputs() & 0x01;
    return 0;
}

// In setup(), before machineOn():
machine.config_HomeController_Callback(myHomeCommand);
```

config_JogController_Callback()

Description

Registers the callback that JOG mode uses to read the jog switches of a Custom Machine. Enter JOG mode afterwards with `beginJog()` / `beginJog(incJog)`.

Syntax

```
machine.config_JogController_Callback(cb);
```

Parameters

- `machine`: a `Machine` object.
- `cb: int cb(jogCommand_t cmd)` – returns the switch state for `READ_JOG_X_POSITIVE`, `READ_JOG_X_NEGATIVE`, `READ_JOG_Y_POSITIVE`, `READ_JOG_Y_NEGATIVE`, `READ_JOG_Z_POSITIVE` or `READ_JOG_Z_NEGATIVE`. Pass `NULL` (default) to remove the callback.

Returns

`bool`: `true` on success; `false` if the machine does not exist, has already been turned on, or is not a Custom Machine.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
int myJogCommand(jogCommand_t feat) {
    if (feat == READ_JOG_X_POSITIVE) return slaveDIO.digitalRead(0);
    if (feat == READ_JOG_X_NEGATIVE) return slaveDIO.digitalRead(1);
    if (feat == READ_JOG_Y_POSITIVE) return slaveDIO.digitalRead(2);
    if (feat == READ_JOG_Y_NEGATIVE) return slaveDIO.digitalRead(3);
    if (feat == READ_JOG_Z_POSITIVE) return slaveDIO.digitalRead(4);
    if (feat == READ_JOG_Z_NEGATIVE) return slaveDIO.digitalRead(5);
    return 0;
}

// In setup(), before machineOn():
machine.config_JogController_Callback(myJogCommand);
```

config_MpgController_Callback()

Description

Registers the callback that MPG mode uses to read the hand-wheel counter of a Custom Machine. Enter MPG mode afterwards with `beginMpg()`.

Syntax

```
machine.config_MpgController_Callback(cb);
```

Parameters

- `machine`: a `Machine` object.
- `cb`: `unsigned long cb(mpgCommand_t cmd)` — returns the raw counter value for `READ_MPG_COUNTER` and the upper bound (wrap value) of the counter for `READ_MPG_RANGE`. Pass `NULL` (default) to remove the callback.

Returns

`bool`: `true` on success; `false` if the machine does not exist, has already been turned on, or is not a Custom Machine.

Make sure the machine exists and configure it **before** calling `machineOn()`.

Example

```
unsigned long myMpgCommand(mpgCommand_t feat) {
  if (feat == READ_MPG_COUNTER) return slaveHID.mpgReadEncoderRaw();
  if (feat == READ_MPG_RANGE)   return UINT32_MAX;
  return 0;
}

// In setup(), before machineOn():
machine.config_MpgController_Callback(myMpgCommand);
```

updateAxes()

Description

Advances a Custom Machine by one interpolation cycle: pushes the new target position of every axis (through `setAxisPosition()` or the `setCB` callback) and reads back the actual positions (through `getAxisPosition()` or `getCB`).

It **must be called at exactly the period** set with `config_CustomActuators_CycleTime()`, normally from the real-time cyclic interrupt of the fieldbus master. `EcatMachine` installs this call automatically.

Syntax

```
machine.updateAxes();
```

Parameters

- `machine`: a `Machine` object.

Returns

None.

Example

```
void MyCyclicCallback() {
    machine.updateAxes();
}

// In setup():
machine.config_CustomActuators_CycleTime(500000);
master.attachRTCyclicCallback(MyCyclicCallback, true);
master.start(500000, ECAT_SYNC);
```

2.3.2 Plugin Override Points

Instead of callbacks, a Custom Machine can be implemented as a **plugin**: a C++ class derived from `Machine` that overrides the virtual hook methods below. `EcatMachine` (Chapter 2.2) is the reference implementation of this pattern.

The plugin remains responsible for calling `updateAxes()` at the period given to `config_CustomActuators_CycleTime()`, typically from the fieldbus master's cyclic interrupt.

Member	Purpose
<code>virtual int homeIoHelper(homeCommand_t cmd)</code>	Return the home-switch state for READ_HOME_X/Y/Z. Called by <code>home()</code> .
<code>virtual int jogIoHelper(jogCommand_t cmd)</code>	Return the jog-switch state for READ_JOG_*_POSITIVE/NEGATIVE. Called in JOG mode.
<code>virtual unsigned long mpgIoHelper(mpgCommand_t cmd)</code>	Return the hand-wheel counter (READ_MPG_COUNTER) or its wrap value (READ_MPG_RANGE). Called in MPG mode.
<code>void setAxisPosition(int axis, long pos) override (protected)</code>	Write the target position (pulses / counts) of axis 0/1/2 to the actuator. Called by <code>updateAxes()</code> .
<code>long getAxisPosition(int axis) override (protected)</code>	Read the actual position (pulses / counts) of axis 0/1/2 from the actuator. Called by <code>updateAxes()</code> .
<code>virtual void machineOn() / machineOff()</code>	Start / stop the fieldbus and enable / disable the drives, then call the base-class method.
<code>virtual ~Machine()</code>	Release plugin resources.
<code>machineMode_t getMachineMode() (protected)</code>	Returns the current mode (e.g. MachineOff); use it to reject configuration after start-up.
<code>bool isHomePrepReady (protected)</code>	Must be set to true by the plugin once the homing hardware is configured; otherwise <code>home()</code> is skipped for safety.

Example

Skeleton of a plugin class:

```
#include "Motion86.h"

class MyMachine : public Machine {
public:
    MyMachine(axisGroup_t g) : Machine(g) { }

    int homeIoHelper(homeCommand_t cmd) override {
        // return the state of the home switch of axis cmd (0/1/2)
        return readMyHomeSwitch(cmd);
    }
    int jogIoHelper(jogCommand_t cmd) override {
        return readMyJogSwitch(cmd);
    }
    unsigned long mpgIoHelper(mpgCommand_t cmd) override {
        if (cmd == READ_MPG_COUNTER) return readMyHandWheel();
        if (cmd == READ_MPG_RANGE)  return UINT32_MAX;
        return 0;
    }
    void machineOn() override {
        config_CustomActuators_CycleTime(myBusCycleTime);
        startMyBus();           // must call updateAxes() every cycle
        isHomePrepReady = true; // homing hardware is ready
        Machine::machineOn();
    }
    void machineOff() override {
        Machine::machineOff();
        stopMyBus();
    }

protected:
    void setAxisPosition(int axis, long pos) override { writeMyTarget(axis, pos); }
    long getAxisPosition(int axis) override           { return readMyActual(axis); }
};

MyMachine machine(CUSTOM_MACHINE_0);
```



Ch. 3

Example

86Duino Coding IDE 501+.

Motion86 Library

3.1 Local Machine

The following sketches are installed with the library (File ► Examples ► Motion86). They drive stepper motors connected to axis group 0 as shown in Chapter 1.4.

3.1.1 Example: Basic Move

Configures a local machine and runs a line / arc path with the native API.

```

/*
  Basic Move

  This demo shows how to control the machine easily by Motion86 Library.
  The path of this example is shown below:
  Start at the '+' on the graph (0, 0), then follow the arrows to move.

      10
      ,----->
      /       |
      |       |
-10 <-----+-----^ 10
           |       |
           |       /
           v-----'
          -10

  Please make sure your machine has enough space to move.
*/

#include "Motion86.h"

// Create Machine object to control machine which you can use any from machine 0 to 2.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // You can also reverse the moving direction if needed.

```

```

// In this case, this machine needs to reverse the direction of the x-axis and y-axis.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// PPU (pulse per unit) is a virtual length unit.
// In this case, it is 80 pulses per millimeter for x-axis.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// We should turn on the machine before control it.
machine.machineOn();
machine.setDefaultFeedrate(400);

// Enable stepper driver.
digitalWrite(EnablePin, LOW);
}

void loop() {
    // Move left to (-10, 0).
    machine.line(-10, 0, 0);

    // Move clockwise to (0, 10) with a radius of 10 in a 90 degree arc.
    machine.arcXY(10, 0, 10, true);

    // Move down to (0, -10).
    machine.line(0, -10, 0);

    // Move counterclockwise to (10, 0) with a radius of 10 in a 90 degree arc.
    machine.arcXY(10, 10, 0, false);

    // Move left to (0, 0).
    machine.line(0, 0, 0);

    // Wait until the movement is finished.
    while (machine.isMoving());
}

```

3.1.2 Example: Basic Move with G-code

The same path as 3.1.1, expressed with `gcode()`.

```

/*
  Basic Move by using Gcode

  This demo shows how to control the machine easily by Motion86 Library.
  The path of this example is shown below:
  Start at the '+' on the graph (0, 0), then follow the arrows to move.

      10
      ,----->
      /       |
      |       |
-10 <-----+-----^ 10
           |       |
           |       /
           v-----'
           -10

  Please make sure your machine has enough space to move.
*/

#include "Motion86.h"

// Create Machine object to control machine which you can use any from machine 0 to 2.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // You can also reverse the moving direction if needed.
  // In this case, this machine needs to reverse the direction of the x-axis and y-axis.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulse per unit) is a virtual length unit.
  // In this case, it is 80 pulses per millimeter for x-axis.

```

```
machine.config_PPU(Axis_X, 80.0);
machine.config_PPU(Axis_Y, 80.0);
machine.config_PPU(Axis_Z, 1600.0);

// We should turn on the machine before control it.
machine.machineOn();

// Enable stepper driver.
digitalWrite(EnablePin, LOW);
}

void loop() {
  // Move left to (-10, 0).
  machine.gcode("G1 X-10 Y0 F400");

  // Move clockwise to (0, 10) with a radius of 10 in a 90 degree arc.
  machine.gcode("G2 X0 Y10 R10 F400");

  // Move down to (0, -10).
  machine.gcode("G1 X0 Y-10 F400");

  // Move counterclockwise to (10, 0) with a radius of 10 in a 90 degree arc.
  machine.gcode("G3 X10 Y0 R10 F400");

  // Move left to (0, 0).
  machine.gcode("G1 X0 Y0 F400");

  // Wait until the movement is finished.
  while (machine.isMoving());
}
```

3.1.3 Example: JOG Control

Manual jog with six push buttons; shows `beginJog()`, `setJogSpeed()`, `setJogOffset()` and `getJogPos()`.

```

/*
  JOG Control
  This example implements the jog control on machine.

  Please make sure your machine has enough space to move.
*/

#include "Motion86.h"

// Create Machine object to control machine which you can use any from machine 0 to 2.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // You can also reverse the moving direction if needed.
  // In this case, this machine needs to reverse the direction of the x-axis and y-axis.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulse per unit) is a virtual length unit.
  // In this case, it is 80 pulses per millimeter for x-axis.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // We should turn on the machine before control it.
  machine.machineOn();

  // Machine is set to jog mode and pins are connected to the buttons to trigger jog.
  machine.beginJog(27, 28, 32, 3, 16, 17, false);

```

```
// IF set machine to Inc Jog mode, you should config jog offset to set how many millimeters
per click.
// machine.beginJog(27, 28, 32, 3, 16, 17, true);
// machine.setJogOffset(1.0);

// Jog mode feedrate.
machine.setJogSpeed(600);

// Enable stepper driver.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Jog position = ");
  Serial.print(machine.getJogPos(AXIS_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(AXIS_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(AXIS_Z));
  delay(500);
}
```

3.1.4 Example: MPG Control

Hand-wheel control with the local encoder Enc3; shows `beginMpg()`, `setMpgAxis()`, `setMpgSpeed()` and `setMpgRatio()`.

```
/*
  MPG Handwheel control
  This example implements the MPG control on machine.

  Please make sure your machine has enough space to move.
*/

#include "Motion86.h"
#include "Encoder.h"

// Create Machine object to control machine which you can use any from machine 0 to 2.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

void setup() {

  while (!Serial);
  // In this case, Handwheel Encoder is installed on encoder 3 and encoder mode is set.
  Enc3.begin(MODE_AB_PHASE);
  pinMode(EnablePin, OUTPUT);

  // You can also reverse the moving direction if needed.
  // In this case, this machine needs to reverse the direction of the x-axis and y-axis.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulse per unit) is a virtual length unit.
  // In this case, it is 80 pulses per millimeter for x-axis.
  machine.config_PPU(AXIS_X, 80.0);
  machine.config_PPU(AXIS_Y, 80.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // We should turn on the machine before control it.
  machine.machineOn();
}
```

```
// Use encoder 3 to trigger Mpg control.
machine.beginMpg(Enc3);

// Choose which axis you want to control by Mpg Hand wheel.
machine.setMpgAxis(Axis_X);
// machine.setMpgAxis(Axis_Y);
// machine.setMpgAxis(Axis_Z);

// Mpg mode feedrate.
machine.setMpgSpeed(600);

// Mpg ratio is the ratio between the hand wheel and the moving distance of machine, it
should more than 0.
machine.setMpgRatio(1);

// Enable stepper driver.
digitalWrite(EnablePin, LOW);
}

void loop() {
  Serial.print("Mpg position = ");
  Serial.print(machine.getJogPos(Axis_X));
  Serial.print(", ");
  Serial.print(machine.getJogPos(Axis_Y));
  Serial.print(", ");
  Serial.println(machine.getJogPos(Axis_Z));
  delay(500);
}
```

3.1.5 Example: Read G-code File

Streams `auto.gcode` from the SD card, throttled with `isCmdBufferFull()`; uses software limits, home switches and `home()`.

```

/*
  Read Gcode File

  This demo shows how the machine reads and runs the gcode file from the SD card
  automatically.

  Please make sure your machine has enough space to move.
  Please make sure the gcode file named as "auto.gcode" is in the SD card.
*/

#include "Motion86.h"
#include "SD.h"

// Create Machine object to control machine which you can use any from machine 0 to 2.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

// The gcode file in the SD card storage.
char *filename = "auto.gcode";
File gfile;
char buf[256];
int ptr;
bool working;

void setup() {
  while (!Serial);
  SD.begin();
  pinMode(EnablePin, OUTPUT);

  // You can also reverse the moving direction if needed.
  // In this case, this machine needs to reverse the direction of the x-axis and y-axis.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

  // PPU (pulse per unit) is a virtual length unit.

```

```

// In this case, it is 80 pulses per millimeter for x-axis.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// Config PosLimit to set the software limit for the machine motion.
machine.config_PosLimit(AXIS_X, 0, 300);
machine.config_PosLimit(AXIS_Y, 0, 200);
machine.config_PosLimit(AXIS_Z, 0, 300);

// Config HomePins to set the hardware limit switch pins.
machine.config_HomePins(2, 7, 8);

// We should turn on the machine before control it.
machine.machineOn();

// Enable software limit to make control safer.
machine.enableSoftLimit();

// Set the feedrate of homing.
machine.setHomeSpeed(1000, 1000, 200);

// Open if there is gcode file in the SD card.
if (SD.exists(filename)) {
    gfile = SD.open(filename);
    working = true;
} else {
    Serial.print("File not exists: ");
    Serial.println(filename);
    while (1);
}

// Enable stepper driver.
digitalWrite(EnablePin, LOW);

// Machine homing.
machine.home();
}

void loop() {
    // Read and process the gcode file from SD card.
    if (working && !machine.isCmdBufferFull()) {

```

```
ptr = 0;
while (gfile.available()) {
  buf[ptr] = gfile.read();
  if (buf[ptr] == '\n') {
    buf[ptr + 1] = '\0';
    machine.gcode(buf);
    break;
  } else {
    ptr ++;
  }
}
if (!gfile.available())
{
  Serial.println("GCODE FINISH");
  gfile.close();
  working = false;
}
}
```

3.1.6 Example: Feedrate Override

Runs a G-code file while an encoder interrupt adjusts `feedrateOverride()` between 0.1 and 2.0.

```

/*
  Feed Rate Override
  This demo shows how to implement feed rate override.
  In this example, the machine runs the gcode file from the SD card automatically,
  and we can change the feed rate by spinning the hand wheel encoder.

  Please make sure your machine has enough space to move.
  Please make sure the gcode file named as "auto.gcode" is in the SD card.
*/

#include "Motion86.h"
#include "Encoder.h"
#include "SD.h"

// Create Machine object to control machine which you can use any from machine 0 to 2.
Machine machine(0);

// Stepper driver enable pin.
int EnablePin = 12;

// The gcode file in the SD card storage.
char *filename = "auto.gcode";
File gfile;
char buf[256];
int ptr;
bool working;

// Set the initial feed rate ratio.
volatile double FeedrateRatio = 1.0;

bool enc_triggered = false;

void encoder_callback(int flag);

void setup() {
  while (!Serial);
  SD.begin();

```

```

// In this case, Handwheel Encoder is installed on encoder 3 and encoder mode is set.
Enc3.begin(MODE_AB_PHASE);
Enc3.setRange(1); //trigger interrupt every time.
Enc3.attachInterrupt(encoder_callback);

pinMode(EnablePin, OUTPUT);

// You can also reverse the moving direction if needed.
// In this case, this machine needs to reverse the direction of the x-axis and y-axis.
machine.config_ReverseDirection(AXIS_X);
machine.config_ReverseDirection(AXIS_Y);

// PPU (pulse per unit) is a virtual length unit.
// In this case, it is 80 pulses per millimeter for x-axis.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// Config PosLimit to set the software limit for the machine motion.
machine.config_PosLimit(AXIS_X, 0, 300);
machine.config_PosLimit(AXIS_Y, 0, 200);
machine.config_PosLimit(AXIS_Z, 0, 300);

// Config HomePins to set the hardware limit switch pins.
machine.config_HomePins(2, 7, 8);

// We should turn on the machine before control it.
machine.machineOn();

// Enable software limit to make control safer.
machine.enableSoftLimit();

// Set the feedrate of homing.
machine.setHomeSpeed(1000, 1000, 200);

// Open if there is gcode file in the SD card.
if (SD.exists(filename)) {
    gfile = SD.open(filename);
    working = true;
} else {
    Serial.print("File not exists: ");
    Serial.println(filename);

```

```

    while (1);
}

// Enable stepper driver.
digitalWrite(EnablePin, LOW);

// Machine homing.
machine.home();
}

void loop() {
    // Read and process the gcode file from SD card.
    if (working && !machine.isCmdBufferFull()) {
        ptr = 0;
        while (gfile.available()) {
            buf[ptr] = gfile.read();
            if (buf[ptr] == '\n') {
                buf[ptr + 1] = '\0';
                machine.gcode(buf);
                break;
            } else {
                ptr++;
            }
        }
        if (!gfile.available())
        {
            Serial.println("GCODE FINISH");
            gfile.close();
            working = false;
        }
    }

    // Override Feedrate when interrupt triggered.
    if (enc_triggered) {
        if (FeedrateRatio > 2.0) FeedrateRatio = 2.0;
        if (FeedrateRatio < 0.1) FeedrateRatio = 0.1;
        machine.feedrateOverride(FeedrateRatio);
        enc_triggered = false;
        Serial.print("FeedrateRatio = ");
        Serial.println(machine.getFeedrateOverride());
    }
}

```

```
void encoder_callback(int flag) {  
    // Feedrate ratio should more than 0.  
    if (flag == INTR_OVERFLOW && FeedrateRatio < 2.0)  
        FeedrateRatio += 0.01;  
    else if (flag == INTR_UNDERFLOW && FeedrateRatio > 0.0)  
        FeedrateRatio -= 0.01;  
    enc_triggered = true;  
}
```

3.1.7 Example: Axis Following

Two local machines; `config_EGearSlave()` makes the slave machine trace the same G-code path scaled per axis (x1.5, x0.5, x1.0).

```

/*
  Axis Following

  This demo shows how to implement the axis following similar to Electronic Gearing
  Application.
  The slave_machine will follow the other machine by the given gear ratio.

  Please make sure your machine has enough space to move.
  Please make sure the gcode file named as "auto.gcode" is in the SD card.
*/

#include "Motion86.h"
#include "SD.h"

// Create Machine object to control machine which you can use any from machine 0 to 2.
// In this case, use machine 0 and machine 1.
Machine machine(0);
Machine slave_machine(1);

// Stepper driver enable pin.
int EnablePin = 12;

// The gcode file in the SD card storage.
char *filename = "auto.gcode";
File gfile;
char buf[256];
int ptr;
bool working;

void setup() {
  while (!Serial);
  pinMode(EnablePin, OUTPUT);

  // You can also reverse the moving direction if needed.
  // In this case, this machine needs to reverse the direction of the x-axis and y-axis.
  machine.config_ReverseDirection(AXIS_X);
  machine.config_ReverseDirection(AXIS_Y);

```

```

// PPU (pulse per unit) is a virtual length unit.
// In this case, it is 80 pulses per millimeter for x-axis.
machine.config_PPU(AXIS_X, 80.0);
machine.config_PPU(AXIS_Y, 80.0);
machine.config_PPU(AXIS_Z, 1600.0);

// Config PosLimit to set the software limit for the machine motion.
machine.config_PosLimit(AXIS_X, 0, 300);
machine.config_PosLimit(AXIS_Y, 0, 200);
machine.config_PosLimit(AXIS_Z, 0, 300);

// Config HomePins to set the hardware limit switch pins.
machine.config_HomePins(2, 7, 8);

// In this case, this slave_machine needs to reverse the direction of the y-axis.
slave_machine.config_ReverseDirection(AXIS_Y);

// Config PPU (pulse per unit) for the slave_machine.
slave_machine.config_PPU(AXIS_X, 80.0);
slave_machine.config_PPU(AXIS_Y, 80.0);
slave_machine.config_PPU(AXIS_Z, 3200.0);

// Config PosLimit to set the software limit for the slave_machine motion.
slave_machine.config_PosLimit(AXIS_X, 0, 200);
slave_machine.config_PosLimit(AXIS_Y, 0, 200);
slave_machine.config_PosLimit(AXIS_Z, 0, 300);

// Config HomePins to set the hardware limit switch pins for slave_machine.
slave_machine.config_HomePins(21, 22, 23);

// Config slave_machine into machine's EGearSlave, it means the slave_machine will
calculate gear ratio and follow the machine's control.
machine.config_EGearSlave(slave_machine, 1.5, 0.5, 1.0);

// We should turn on the machine before control it.
machine.machineOn();

// Enable software limit to make control safer.
machine.enableSoftLimit();

// Set the feedrate of homing.

```

```

machine.setHomeSpeed(1000, 1000, 200);
slave_machine.setHomeSpeed(1000, 1000, 100);

// Open if there is gcode file in the SD card.
if (SD.exists(filename)) {
    gfile = SD.open(filename);
    working = true;
} else {
    Serial.print("File not exists: ");
    Serial.println(filename);
    while (1);
}

// Enable stepper driver.
digitalWrite(EnablePin, LOW);

// Machine homing.
machine.home();
}

void loop() {
    // Read and process the gcode file from SD card
    // slave_machine will zoom gcode path more wider (x1.5) and shorter (x0.5).
    if (working && !machine.isCmdBufferFull()) {
        ptr = 0;
        while (gfile.available()) {
            buf[ptr] = gfile.read();
            if (buf[ptr] == '\n') {
                buf[ptr + 1] = '\0';
                machine.gcode(buf);
                break;
            } else {
                ptr ++;
            }
        }
        if (!gfile.available())
        {
            Serial.println("GCODE FINISH");
            gfile.close();
            working = false;
        }
    }
}

```

}

3.1.8 Example: Multiple Machines

Serial command line (<group> INIT / CLOSE / HOME / EMG / ~EMG / POS / PIN / GFILE <file> / <gcode>) managing up to three machines created dynamically with new Machine(group).

```

/*
Multiple Machines
This demo demonstrates how to config machine from serial monitor command line.
The instruction and the example of command format are shown as below.

Instruction:
1. open serial monitor.
2. First parameter should be machine group id.
3. Second parameter can be Init, Close, Home, Pos, Pin, Gfile or Gcode command.
4. If you use Gfile to read file, you should add third parameter as gcode filename in SD
card.

Example:
0 INIT
1 INIT
0 HOME
1 HOME
0 GFILE auto.gcode
1 GFILE auto2.gcode

Please make sure your machine has enough space to move.
Please make sure the gcode file named as "auto.gcode" is in the SD card.
*/

#include "Motion86.h"
#include "SD.h"

// Array of pointer to Machine Objects and you can have up to 3 machines.
Machine *machine[3];

// Stepper driver enable pin.
int EnablePin = 12;

// The specific gcode files for each machine in the SD card storage.
File gfile[3];
bool fileopened[3] = {false, false, false};

```

```

char *filename;
char buf[256];
int ptr;

double ActualPos[3];

// PPU (pulse per unit) is a virtual length unit.
// Set the pulses per millimeter for each axis of each machine.
double AxisPPU[3][3] = {{80.0, 80.0, 1600.0},
                        {80.0, 80.0, 3200.0},
                        {80.0, 80.0, 3200.0}};

// You can also reverse the moving direction if needed.
// Set 1 to reverse and 0 for not to reverse for each axis of each machine.
int MotorDirInvert[3][3] = {{1, 1, 0},
                           {0, 1, 0},
                           {0, 0, 1}};

// Set the minimum software position limit for each axis of each machine.
int MotorPosLimitMin[3][3] = {{0, 0, 0},
                              {0, 0, 0},
                              {0, 0, 0}};

// Set the maximum software position limit for each axis of each machine.
int MotorPosLimitMax[3][3] = {{300, 200, 300},
                              {200, 200, 300},
                              {200, 200, 300}};

// Set the hardware limit switch pins for each axis of each machine.
int Limit_PinMap[3][3] = {{ 2,  7,  8},
                          {21, 22, 23},
                          {24, 25, 26}};

void processingCMD(int group, char *command);

void setup() {
    while (!Serial);
    SD.begin();
    pinMode(EnablePin, OUTPUT);
    // Enable stepper driver.
    digitalWrite(EnablePin, LOW);
}

```

```

void loop() {
  ptr = 0;
  // Read and process if receive command from serial.
  while (Serial.available() > 0) {
    buf[ptr] = (char)Serial.read();
    if (buf[ptr] == '\n') {
      buf[ptr] = '\0';
      Serial.println(buf);
      processingCMD(atoi(buf), &buf[2]);
      break;
    } else {
      ptr ++;
    }
  }

  // Check every machine if they are valid or not.
  for (int i = 0; i < 3; i++) {
    if (machine[i] == NULL) continue;
    if (!machine[i]->isValid()) continue;
    // Read and process the gcode file from SD card
    if (fileopened[i] && !machine[i]->isCmdBufferFull())
    {
      ptr = 0;
      while (gfile[i].available()) {
        buf[ptr] = gfile[i].read();
        if (buf[ptr] == '\n') {
          buf[ptr + 1] = '\0';
          machine[i]->gcode(buf);
          break;
        } else {
          ptr ++;
        }
      }
      if (!gfile[i].available()) {
        Serial.println("GCODE FINISH");
        gfile[i].close();
        fileopened[i] = false;
      }
    }
  }
}

```

```

void processingCMD(int group, char *command) {
    if (group < 0 || group > 2) return;
    if (strcmp(command, "INIT") == 0) {
        if (machine[group] != NULL) return;
        // Construct a new machine.
        machine[group] = new Machine(group);
        // Reverse the moving direction for each axis of each machine if needed.
        if (MotorDirInvert[group][AXIS_X]) machine[group]->config_ReverseDirection(AXIS_X);
        if (MotorDirInvert[group][AXIS_Y]) machine[group]->config_ReverseDirection(AXIS_Y);
        if (MotorDirInvert[group][AXIS_Z]) machine[group]->config_ReverseDirection(AXIS_Z);
        // Set the pulses per millimeter for each axis of each machine.
        machine[group]->config_PPU(AXIS_X, AxisPPU[group][AXIS_X]);
        machine[group]->config_PPU(AXIS_Y, AxisPPU[group][AXIS_Y]);
        machine[group]->config_PPU(AXIS_Z, AxisPPU[group][AXIS_Z]);
        // Set the software position limit for each axis of each machine.
        machine[group]->config_PosLimit(AXIS_X, MotorPosLimitMin[group][AXIS_X],
MotorPosLimitMax[group][AXIS_X]);
        machine[group]->config_PosLimit(AXIS_Y, MotorPosLimitMin[group][AXIS_Y],
MotorPosLimitMax[group][AXIS_Y]);
        machine[group]->config_PosLimit(AXIS_Z, MotorPosLimitMin[group][AXIS_Z],
MotorPosLimitMax[group][AXIS_Z]);
        // Set the hardware limit switch pins for each axis of each machine.
        machine[group]->config_HomePins(Limit_PinMap[group][AXIS_X], Limit_PinMap[group][AXIS_Y],
Limit_PinMap[group][AXIS_Z]);
        // We should turn on the machine before control it.
        machine[group]->machineOn();
        return;
    }

    if (machine[group] == NULL) return;
    if (!machine[group]->isValid()) return;

    if (strcmp(command, "CLOSE") == 0) {
        // Deconstructor.
        delete machine[group];
        machine[group] = NULL;
    } else if (strcmp(command, "HOME") == 0) {
        // Set the feedrate of homing.
        machine[group]->setHomeSpeed(1000, 1000, 200);
        // Machine homing.
        machine[group]->home();
    }
}

```

```

} else if (strcmp(command, "EMG") == 0) {
    // Emergency stop.
    machine[group]->emgStop();
    if (fileopened[group]) {
        gfile[group].close();
        fileopened[group] = false;
    }
} else if (strcmp(command, "~EMG") == 0) {
    // Clear emergency stop status.
    machine[group]->clearEMGStop();
} else if (strcmp(command, "POS") == 0) {
    // Get actual position.
    machine[group]->getActualPos(ActualPos[AXIS_X], ActualPos[AXIS_Y], ActualPos[AXIS_Z]);
    Serial.print("X:");
    Serial.print(ActualPos[AXIS_X]);
    Serial.print(", Y:");
    Serial.print(ActualPos[AXIS_Y]);
    Serial.print(", Z:");
    Serial.println(ActualPos[AXIS_Z]);
} else if (strcmp(command, "PIN") == 0) {
    // Get status of hardware limit switches.
    Serial.println(digitalRead(Limit_PinMap[group][AXIS_X]));
    Serial.println(digitalRead(Limit_PinMap[group][AXIS_Y]));
    Serial.println(digitalRead(Limit_PinMap[group][AXIS_Z]));
} else if (strcmp(command, "GFILE") == 32) {
    // Decode parameters to open the gcode file.
    filename = &command[6];
    if (fileopened[group]) {
        Serial.println("Already running.");
    } else if (SD.exists(filename)) {
        gfile[group] = SD.open(filename);
        fileopened[group] = true;
    } else {
        Serial.print("File not exists: ");
        Serial.println(filename);
    }
} else {
    // Other commands are treated as gcode.
    machine[group]->gcode(command);
}
}

```

3.2 EcatMachine

EtherCAT servo control through 86EVA. Slave 0 is a QEC-RXXMP3S configured in 86EVA as VirtualServo1~VirtualServo3.

3.2.1 Example: Basic Move with EtherCAT Servos

The same path as 3.1.1, driven by three EtherCAT servos. Note the include order and that no EtherCAT code is required.

```
#include "myeva.h"
#include "Motion86_Ethercat.h"

// EcatMachine only accepts CUSTOM_MACHINE_0 ~ CUSTOM_MACHINE_2.
EcatMachine machine(CUSTOM_MACHINE_0);

void setup() {
  // Start the EtherCAT master and the 86EVA virtual devices.
  EVA.begin();

  // Bind the virtual servos to the axes (before machineOn()).
  machine.attach_EcatServo_X(VirtualServo1);
  machine.attach_EcatServo_Y(VirtualServo2);
  machine.attach_EcatServo_Z(VirtualServo3);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);

  // PPU: encoder counts per mm.
  machine.config_PPU(AXIS_X, 1600.0);
  machine.config_PPU(AXIS_Y, 1600.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // machineOn() initialises the EtherCAT drives and switches them to CSP mode.
  machine.machineOn();
  machine.setDefaultFeedrate(400);
}

void loop() {
  // Move left to (-10, 0).
  machine.line(-10, 0, 0);
}
```

```
// Clockwise 90-degree arc with radius 10 to (0, 10).
machine.arcXY(10, 0, 10, true);
// Move down to (0, -10).
machine.line(0, -10, 0);
// Counter-clockwise 90-degree arc with radius 10 to (10, 0).
machine.arcXY(10, 10, 0, false);
// Move back to (0, 0).
machine.line(0, 0, 0);
// Wait until the planned motion is finished.
while (machine.isMoving());
}
```

3.2.2 Example: EtherCAT Homing, Jog and Hand Wheel

Declares home switches, jog buttons and a hand wheel on the Virtual Arduino, homes the machine, and then lets the operator switch between jog and MPG mode.

```
#include "myeva.h"
#include "Motion86_Ethercat.h"

// EcatMachine only accepts CUSTOM_MACHINE_0 ~ CUSTOM_MACHINE_2.
EcatMachine machine(CUSTOM_MACHINE_0);

void setup() {
  // Start the EtherCAT master and the 86EVA virtual devices.
  EVA.begin();

  // Bind the virtual servos to the axes (before machineOn()).
  machine.attach_EcatServo_X(VirtualServo1);
  machine.attach_EcatServo_Y(VirtualServo2);
  machine.attach_EcatServo_Z(VirtualServo3);

  // Virtual Arduino inputs: home switches 0~2, jog buttons 4~9 (active low).
  machine.attach_EcatHomePin_X(0);
  machine.attach_EcatHomePin_Y(1);
  machine.attach_EcatHomePin_Z(2);
  machine.attach_EcatJogPins_X(4, 5, ACTIVE_LOW, ACTIVE_LOW);
  machine.attach_EcatJogPins_Y(6, 7, ACTIVE_LOW, ACTIVE_LOW);
  machine.attach_EcatJogPins_Z(8, 9, ACTIVE_LOW, ACTIVE_LOW);
  machine.attach_EcatMpg(VirtualMpg1);

  // Reverse the direction of an axis if required.
  machine.config_ReverseDirection(AXIS_X);

  // PPU: encoder counts per mm.
  machine.config_PPU(AXIS_X, 1600.0);
  machine.config_PPU(AXIS_Y, 1600.0);
  machine.config_PPU(AXIS_Z, 1600.0);

  // machineOn() initialises the EtherCAT drives and switches them to CSP mode.
  machine.machineOn();
  machine.setDefaultFeedrate(400);

  machine.setHomeSpeed(20, 20, 10);
```

```
machine.home();
while (machine.isMoving());
}

void loop() {
  // Jog for 30 seconds ...
  machine.beginJog();
  machine.setJogSpeed(100);
  delay(30000);
  machine.endJog();

  // ... then hand wheel on X for 30 seconds.
  machine.beginMpg();
  machine.setMpgAxis(AXIS_X);
  machine.setMpgSpeed(100);
  machine.setMpgRatio(0.01);
  delay(30000);
  machine.endMpg();
}
```

3.3 Custom Machine

A Custom Machine driven through callbacks and the generic 86Duino EtherCAT library (Ethercat.h), without 86EVA. The user code owns the EtherCAT master and calls `updateAxes()` from the master's real-time cyclic callback.

3.3.1 Example: Custom Machine with Callbacks

```
#include "Motion86.h"
#include "Ethercat.h"

// A Machine on a custom axis group, driven through callbacks.
Machine machine(CUSTOM_MACHINE_0);
EthercatMaster master;
EthercatDevice_CiA402 servo[3];

// Position callbacks: axis is 0/1/2 for X/Y/Z, positions are in pulses / counts.
void setRemoteTargetPos(int axis, long pos) { servo[axis].setTargetPosition(pos); }
long getRemoteTargetPos(int axis)          { return servo[axis].getPositionActualValue(); }

// Home switch callback: digital input 0 of each drive.
int myHomeCommand(homeCommand_t feat) {
    if (feat == READ_HOME_X) return servo[0].getDigitalInputs() & 0x01;
    if (feat == READ_HOME_Y) return servo[1].getDigitalInputs() & 0x01;
    if (feat == READ_HOME_Z) return servo[2].getDigitalInputs() & 0x01;
    return 0;
}

// Called by the EtherCAT master every cycle (500 us).
void MyCyclicCallback() {
    machine.updateAxes();
}

void setup() {
    master.begin();
    for (int i = 0; i < 3; i++) {
        servo[i].attach(0, i, master);
        servo[i].setDc(500000);
    }
}
```

```

machine.config_ReverseDirection(AXIS_X);
machine.config_PPU(AXIS_X, 1600.0);
machine.config_PPU(AXIS_Y, 1600.0);
machine.config_PPU(AXIS_Z, 1600.0);

// Custom actuator hooks: cycle time must match the master cycle.
machine.config_CustomActuators_CycleTime(500000);
machine.config_CustomActuators_Callbacks(setRemoteTargetPos, getRemoteTargetPos);
machine.config_HomeController_Callback(myHomeCommand);

master.attachRTCCyclicCallback(MyCyclicCallback, true);
master.start(500000, ECAT_SYNC);
for (int i = 0; i < 3; i++) {
    servo[i].enable();
    servo[i].setCiA402Mode(CIA402_CSP_MODE);
}

// The machine must be turned on before it can be controlled.
machine.machineOn();
machine.setDefaultFeedrate(400);
}

void loop() {
    // Move left to (-10, 0).
    machine.line(-10, 0, 0);
    // Clockwise 90-degree arc with radius 10 to (0, 10).
    machine.arcXY(10, 0, 10, true);
    // Move down to (0, -10).
    machine.line(0, -10, 0);
    // Counter-clockwise 90-degree arc with radius 10 to (10, 0).
    machine.arcXY(10, 10, 0, false);
    // Move back to (0, 0).
    machine.line(0, 0, 0);
    // Wait until the planned motion is finished.
    while (machine.isMoving());
}

```



Appendix

A.1 G-code Command Reference

Description

Machine::gcode() accepts one command per call. In the command formats below, a "-" after a word (e.g. G1 Axes- F-) stands for a numeric value. *Axes* means any of X- Y- Z-; *Offsets* means any of I- J- K- (arc center, relative to the start point by default).

G-code	Description
G1	Linear motion
G2	Clockwise circular / helical motion
G3	Counter-clockwise circular / helical motion
G4	Dwell (pause)
G17	Select the XY plane for circular motion (default)
G18	Select the XZ plane
G19	Select the YZ plane
G90	Absolute positioning (default)
G91	Relative (incremental) positioning
G90.1	Arc center given in absolute coordinates
G91.1	Arc center relative to the start point (default)
G92	Coordinate system offset (set the current position)

G1

G1 Axes- F-

Linear motion. G90 / G91 select absolute or relative coordinates.

- Axes: end point (absolute under G90, incremental under G91).
- F: feedrate.

Usage examples

```
// 1. Absolute move to (10, 10, 20) at F500
machine.gcode("G90");
machine.gcode("G1 X10 Y10 Z20 F500");

// 2. Relative move by (+10, -10, +20)
machine.gcode("G91");
machine.gcode("G1 X10 Y-10 Z20 F500");

// 3. Only Z moves to 50; X and Y stay unchanged
machine.gcode("G90");
machine.gcode("G1 Z50 F500");
```

G2, G3

Center mode: G2 Axes- Offsets- P- F-

Radius mode: G2 Axes- R- P- F-

G2 is clockwise, G3 counter-clockwise. The plane of the arc / helix is selected with G17, G18, G19. Depending on the parameter combination, arcs, full circles and helices can be produced; see 1.5 Motion Methods.

- **Axes**: end point; include the third axis to obtain a helix.
- **Offsets**: I J K – center of the arc.
- **R**: radius; a negative value selects the arc greater than 180°.
- **P**: number of full turns (circles or helix pitches).
- **F**: feedrate.

Usage examples

```
// 1. 90-degree CW arc (0,0,0) -> (50,50,0) around center (50,0,0), center mode
machine.gcode("G90");
machine.gcode("G1 X0 Y0 Z0 F500");
machine.gcode("G2 X50 Y50 I50 J0 F500");
// 2. Same arc in radius mode
machine.gcode("G90");
machine.gcode("G1 X0 Y0 Z0 F500");
machine.gcode("G2 X50 Y50 R50 F500");
// 3. 270-degree CCW arc to the same end point, center (50,0,0)
machine.gcode("G90");
machine.gcode("G1 X0 Y0 Z0 F500");
machine.gcode("G3 X50 Y50 R-50 F500");
// 4. 90-degree CCW arc around center (0,50,0)
machine.gcode("G90");
machine.gcode("G1 X0 Y0 Z0 F500");
machine.gcode("G3 X50 Y50 R50 F500");
// 5. Full CW circle in the XZ plane around center (0,0,50)
machine.gcode("G90");
machine.gcode("G1 X0 Y0 Z0 F500");
machine.gcode("G18");
machine.gcode("G2 X0 Z0 I0 K50 F500");
// 6. CW helix, XY plane, center at the origin, 10 turns from (50,0,0) to (50,0,100)
machine.gcode("G90");
machine.gcode("G1 X50 Y0 Z0 F500");
machine.gcode("G17");
machine.gcode("G2 X50 Y0 Z100 I-50 J0 P10 F500");
```

G4

G4 S-

G4 P-

Pauses the machine for the given time.

- S: dwell time in seconds (integer).
- P: dwell time in milliseconds.

Usage examples

```
// 1. Wait 2 seconds at (50,0,0), then move to (0,50,0)
machine.gcode("G90");
machine.gcode("G1 X50 Y0 Z0 F500");
machine.gcode("G4 S2");
machine.gcode("G1 X0 Y50 Z0 F500");

// 2. Wait 0.5 seconds
machine.gcode("G4 P500");
```

G17, G18, G19

G17

G18

G19

Selects the plane for circular and helical motion (G2 / G3).

- G17: XY plane (default).
- G18: XZ plane.
- G19: YZ plane.

Usage examples

```
// 1. CW circle R50 in XY, then CCW circle R50 in YZ, both from the origin
machine.gcode("G90");
machine.gcode("G1 X0 Y0 Z0 F500");
machine.gcode("G17");
machine.gcode("G2 X0 Y0 I50 J0 F500");
machine.gcode("G19");
machine.gcode("G3 Y0 Z0 J0 K50 F500");

// 2. CW helix in XZ, center (0,0,50), Y as height, 10 turns, (0,0,0) -> (0,100,0)
machine.gcode("G90");
machine.gcode("G1 X0 Y0 Z0 F500");
machine.gcode("G18");
machine.gcode("G2 X0 Y100 Z0 I0 K50 P10 F500");
```

G90, G91

G90

G91

Sets the positioning mode.

- **G90**: absolute mode (default): *Axes* of **G1**, **G2**, **G3** are absolute positions.
- **G91**: relative mode: *Axes* are increments from the current position.

Usage examples

```
// Both sequences draw the same CW arc from (50,0,0) to (150,0,0) around (100,0,0)

// 1. Absolute end point
machine.gcode("G90");
machine.gcode("G1 X50 Y0 Z0 F500");
machine.gcode("G2 X150 Y0 I50 J0 F500");

// 2. Incremental end point
machine.gcode("G90");
machine.gcode("G1 X50 Y0 Z0 F500");
machine.gcode("G91");
machine.gcode("G2 X100 Y0 I50 J0 F500");
```

G90.1, G91.1

G90.1

G91.1

Sets the arc-center mode.

- **G90.1**: absolute center mode: *Offsets* of **G2**, **G3** are absolute positions.
- **G91.1**: relative center mode (default): *Offsets* are relative to the start point.

Usage examples

```
// Both sequences draw the same CW arc from (50,0,0) to (150,0,0) around (100,0,0)

// 1. Absolute center
machine.gcode("G90");
machine.gcode("G1 X50 Y0 Z0 F500");
machine.gcode("G90.1");
machine.gcode("G2 X150 Y0 I100 J0 F500");

// 2. Relative center
machine.gcode("G90");
machine.gcode("G1 X50 Y0 Z0 F500");
machine.gcode("G91.1");
machine.gcode("G2 X150 Y0 I50 J0 F500");
```

G92

G92 Axes -

Coordinate system offset: declares the given coordinates to be the current position.

- **Axes**: coordinates to assign to the current position.

Usage examples

```
// The current position becomes the origin of the work coordinate system  
machine.gcode("G92 X0 Y0 Z0");
```

A.2 Enumerations

Types defined in `Motion86.h` (and `pulseMode_t` in `motion/MotInterface.h`).

Type / Constant	Value	Description
<code>machineAxis_t</code>		Axis index
<code>AXIS_X</code>	0	X axis
<code>AXIS_Y</code>	1	Y axis
<code>AXIS_Z</code>	2	Z axis
<code>axisGroup_t</code>		Axis group (controller instance)
<code>MACHINE_UNUSED</code>	-1	Returned when <code>machineOn()</code> fails
<code>MACHINE_0 / 1 / 2</code>	0 / 1 / 2	Local machines
<code>CUSTOM_MACHINE_0 / 1 / 2</code>	100 / 101 / 102	Custom machines (EtherCAT, Modbus, ...)
<code>ioPolarity_t</code>		Logic polarity of an input
<code>ACTIVE_LOW</code>	0	Active when low (normally closed)
<code>ACTIVE_HIGH</code>	1	Active when high (normally open)
<code>homeCommand_t</code>		Home-controller callback command
<code>READ_HOME_X / Y / Z</code>	0 / 1 / 2	Read the home switch of the axis
<code>jogCommand_t</code>		Jog-controller callback command
<code>READ_JOG_X_POSITIVE / READ_JOG_X_NEGATIVE</code>	0 / 1	Read the X+ / X- jog switch
<code>READ_JOG_Y_POSITIVE / READ_JOG_Y_NEGATIVE</code>	2 / 3	Read the Y+ / Y- jog switch
<code>READ_JOG_Z_POSITIVE / READ_JOG_Z_NEGATIVE</code>	4 / 5	Read the Z+ / Z- jog switch
<code>mpgCommand_t</code>		MPG-controller callback command
<code>READ_MPG_COUNTER</code>	0	Read the raw hand-wheel counter
<code>READ_MPG_RANGE</code>	1	Read the upper bound (wrap value) of the counter
<code>pulseMode_t</code>		Pulse output mode
<code>MODE_STEP_DIR</code>	0	Step (pulse) and Direction
<code>MODE_CWCCW</code>	1	CW and CCW pulses
<code>MODE_AB_PHASE</code>	2	A/B phase quadrature
<code>MODE_CAPTURE</code>	3	Position capture mode
<code>MODE_SSI</code>	4	Synchronous Serial Interface (continuous mode, no interrupt)

A.3 Usage Restrictions

- **Timer conflict.** Motion86 uses the hardware timer. It cannot be used together with the **TimerWDT** or **AI Servo86** libraries.
- **One instance per axis group.** Constructing a second **Machine** on a group that is already in use yields an invalid object (`isValid()` returns `false`).
- **Configure first, then machineOn().** `config_*` and `attach_*` calls made after `machineOn()` are rejected and return `false`.
- **Local-only methods.** `config_MaxPulseSpeed()` has no effect on a Custom Machine.
- **Custom Machine cycle time.** `updateAxes()` must run at exactly the period given to `config_CustomActuators_CycleTime()`, from a real-time context.
- **EtherCAT include order.** `myeva.h` must be included before `Motion86_Ethercat.h`.
- **Homing on a Custom Machine** requires the home hardware to be declared (`attach_EcatHomePin_*`), a home callback, or the plugin setting `isHomePrepReady = true`; otherwise `home()` is skipped.
- **Arc geometry.** Center / radius-mode arcs whose end point is not on the circle produce undefined motion.
- **Command buffer.** 1024 blocks. Use `isCmdBufferFull()` or `getFreeCmdCount()` to throttle when streaming G-code.

Warranty

This product is warranted to be in good working order for a period of one year from the date of purchase. Should this product fail to be in good working order at any time during this period, we will, at our option, replace or repair it at no additional charge except as set forth in the following terms. This warranty does not apply to products damaged by misuse, modifications, accident or disaster. Vendor assumes no liability for any damages, lost profits, lost savings or any other incidental or consequential damage resulting from the use, misuse of, originality to use this product. Vendor will not be liable for any claim made by any other related party. Return authorization must be obtained from the vendor before returned merchandise will be accepted. Authorization can be obtained by calling or faxing the vendor and requesting a Return Merchandise Authorization (RMA) number. Returned goods should always be accompanied by a clear problem description.

All Trademarks appearing in this manuscript are registered trademark of their respective owners. All Specifications are subject to change without notice.

©ICOP Technology Inc. 2026