# StepperMC

Generated by Doxygen 1.9.6

# Chapter 1

# Stepper MC

A sophisticated Stepper Library with extensive Motion Control functions.

Stepper MC supports accelerations ramps with optimized timing due to minmum computational overhead, backlash handling and motion in engineering units. Steppers can be configured either in unlimited, limited or modulo mode. Feed constant for units per turn can be set individually.

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 StepperMC Class Reference

### Public Member Functions

- StepperMC (uint8_t pin1, uint8_t pin2, uint8_t pin3, uint8_t pin4, uint16_t steps=4096)

  *Constructor, connect 4 phase Stepper to Arduino pins.*
- StepperMC (uint8_t pinDir, uint8_t pinStep, uint16_t steps=4096)

  *Constructor, connect Stepper with directon and step lines.*
- void handle ()

  *Realtime handle. Performs one motor step when necessary. Call in fastest loop.*
- void setIncrements (int32_t pos)

  *Set target position in increments.*
- void setIncrementsRelative (int32_t steps)

  *Set target position relative to current target position in increments.*
- int32_t getIncrements ()

  *Get current position in increments.*
- void setPosition (float pos)

  *Set target position in engineering units.*
- void setPositionRelative (float pos)

  *Set target position relative to current target position in engineering units.*
- float getPosition ()

  *Get current position in engineering units.*
- void moveTarget ()

  *Call handle cyclic until target position reached (blocking)*
- bool inTarget ()

  *Check whether target position is reached.*
- void stop ()

  *Set new target position for fast stop from current postion.*
- void setZero ()

  *Set the current Position as zero. Use only on standstill.*
- void setSpeed (uint16_t freqMax, uint16_t acc=0)

  *Set speed and optional acceleration for motion commands.*
- void adjustZero (int32_t steps)

  *Adjust the zero position by some increments (steps)*
- void setBacklash (int32_t steps)

*Set backlash of drive. Motion within backlash range is not counted for actual position.*

- void setGearRatio (int32_t motor, int32_t load)

  *Set gear ratio between motor and load. Used for motion in engineering units.*
- void setFeedConst (float feed)

  *Set feed constant for motion in engineering units.*
- void setModulo (uint16_t steps=0)

  *Set modulo range for motion. After the modulo distance possition repeats. Useful e.g. for repeating 360° axis.*
- void setUnlimited ()

  *Remove any limits and modulo settings for axis (default mode)*
- void setPositionLimit (float lower, float upper)

  *Set position limits in engineering units and make axis limited. Target positions beyond positive or negative limit are truncated to the limit.*
- void reverseDir (bool neg)

  *Reverse motion direction of axis on low level.*
- void setPowersaveTime (uint16_t seconds)

  *Set timer for powersave mode of axis.*

## 4.1.1 Detailed Description

Definition at line 5 of file StepperMC.h.

## 4.1.2 Constructor & Destructor Documentation

### 4.1.2.1 StepperMC() [1/2]

```
StepperMC::StepperMC (
             uint8_t pin1,
             uint8_t pin2,
             uint8_t pin3,
             uint8_t pin4,
             uint16_t steps = 4096 )
```

Constructor, connect 4 phase Stepper to Arduino pins.

**Parameters**

| | |
|---|---|
| *pin1* | Pin for Phase 1 |
| *pin2* | Pin for Phase 2 |
| *pin3* | Pin for Phase 3 |
| *pin4* | Pin for Phase 4 |
| *steps* | Steps per motor revolution (optional, default = 4096) |

Definition at line 18 of file StepperMC.cpp.

#### 4.1.2.2 StepperMC() [2/2]

```
StepperMC::StepperMC (
            uint8_t pinDir,
            uint8_t pinStep,
            uint16_t steps = 4096 )
```

Constructor, connect Stepper with directon and step lines.

**Parameters**

| | |
|---|---|
| *pinDir* | Pin for motor direction |
| *pinStep* | Pin to execute a step |
| *steps* | Steps per motor revolution (optional, default = 4096) |

Definition at line 39 of file StepperMC.cpp.

### 4.1.3 Member Function Documentation

#### 4.1.3.1 adjustZero()

```
void StepperMC::adjustZero (
            int32_t steps )
```

Adjust the zero position by some increments (steps)

**Parameters**

| | |
|---|---|
| *steps* | Number of steps for adjustment |

Definition at line 315 of file StepperMC.cpp.

#### 4.1.3.2 getIncrements()

```
int32_t StepperMC::getIncrements ( )
```

Get current position in increments.

**Returns**

Current postion in inkrements (steps)

Definition at line 269 of file StepperMC.cpp.

### 4.1.3.3 getPosition()

```
float StepperMC::getPosition ( )
```

Get current position in engineering units.

**Returns**

Current postion in engineering units

Definition at line 275 of file StepperMC.cpp.

### 4.1.3.4 handle()

```
void StepperMC::handle ( )
```

Realtime handle. Performs one motor step when necessary. Call in fastest loop.

Definition at line 83 of file StepperMC.cpp.

### 4.1.3.5 inTarget()

```
bool StepperMC::inTarget ( )
```

Check whether target position is reached.

**Returns**

true = target position reached, false = target not reached

Definition at line 281 of file StepperMC.cpp.

### 4.1.3.6 moveTarget()

```
void StepperMC::moveTarget ( )
```

Call handle cyclic until target position reached (blocking)

Definition at line 299 of file StepperMC.cpp.

### 4.1.3.7 reverseDir()

```
void StepperMC::reverseDir (
             bool neg )
```

Reverse motion direction of axis on low level.

**Parameters**

| | |
|---|---|
| *neg* | true: invert motion direction, false: do not invert |

Definition at line 396 of file StepperMC.cpp.

#### 4.1.3.8 setBacklash()

```
void StepperMC::setBacklash (
             int32_t steps )
```

Set backlash of drive. Motion within backlash range is not counted for actual position.

**Parameters**

| | |
|---|---|
| *steps* | Basklash range in incremets (steps) |

Definition at line 321 of file StepperMC.cpp.

#### 4.1.3.9 setFeedConst()

```
void StepperMC::setFeedConst (
             float feed )
```

Set feed constant for motion in engineering units.

**Parameters**

| | |
|---|---|
| *feed* | One load side revolution in engineering units (default: 360) |

Definition at line 354 of file StepperMC.cpp.

#### 4.1.3.10 setGearRatio()

```
void StepperMC::setGearRatio (
             int32_t motor,
             int32_t load )
```

Set gear ratio between motor and load. Used for motion in engineering units.

**Parameters**

| | |
|---|---|
| *motor* | Number of gear teeth on motor side (default 1) |
| *load* | Number of gear teeth on load side (default 1) |

Definition at line 348 of file StepperMC.cpp.

#### 4.1.3.11 setIncrements()

```
void StepperMC::setIncrements (
             int32_t pos )
```

Set target position in increments.

**Parameters**

| | |
|---|---|
| *pos* | Absolute target postion in inkrements (steps) |

Definition at line 206 of file StepperMC.cpp.

#### 4.1.3.12 setIncrementsRelative()

```
void StepperMC::setIncrementsRelative (
             int32_t steps )
```

Set target position relative to current target position in increments.

**Parameters**

| | |
|---|---|
| *steps* | Relative target postion in inkrements (steps) |

Definition at line 217 of file StepperMC.cpp.

#### 4.1.3.13 setModulo()

```
void StepperMC::setModulo (
             uint16_t steps = 0 )
```

Set modulo range for motion. After the modulo distance possition repeats. Useful e.g. for repeating 360° axis.

**Parameters**

| | |
|---|---|
| *steps* | Number of increments (steps) after which position is reset. steps = 0 takes one turn on liad side. Take care with uneven gear ratios leading to fractional increments, they lead to inaccuracies. |

Definition at line 362 of file StepperMC.cpp.

#### 4.1.3.14 setPosition()

```
void StepperMC::setPosition (
             float pos )
```

Set target position in engineering units.

**Parameters**

| | |
|---|---|
| *pos* | Absolute target position in engineering units |

Definition at line 223 of file StepperMC.cpp.

#### 4.1.3.15 setPositionLimit()

```
void StepperMC::setPositionLimit (
             float lower,
             float upper )
```

Set position limits in engineering units and make axis limited. Target positions beyond positive or negative limit are truncated to the limit.

**Parameters**

| | |
|---|---|
| *lower* | Lower position limit in engineering units |
| *upper* | Upper position limit in engineering units |

Definition at line 387 of file StepperMC.cpp.

#### 4.1.3.16 setPositionRelative()

```
void StepperMC::setPositionRelative (
             float pos )
```

Set target position relative to current target position in engineering units.

**Parameters**

| | |
|---|---|
| *pos* | Relative target postion in engineering units |

Definition at line 229 of file StepperMC.cpp.

### 4.1.3.17 setPowersaveTime()

```
void StepperMC::setPowersaveTime (
             uint16_t seconds )
```

Set timer for powersave mode of axis.

**Parameters**

| | |
|---|---|
| *seconds* | Number of seconds after which the amplifiers are switched to idle. |

Definition at line 401 of file StepperMC.cpp.

### 4.1.3.18 setSpeed()

```
void StepperMC::setSpeed (
             uint16_t freqMax,
             uint16_t acc = 0 )
```

Set speed and optional acceleration for motion commands.

**Parameters**

| | |
|---|---|
| *freqMax* | Maximum speed as increments (steps) per second |
| *acc* | Optional acceleration for speed ramps. acc = 0 means constant speed and reduces computing time of handle() significantly. |

Definition at line 328 of file StepperMC.cpp.

### 4.1.3.19 setUnlimited()

```
void StepperMC::setUnlimited ( )
```

Remove any limits and modulo settings for axis (default mode)

Definition at line 377 of file StepperMC.cpp.

#### 4.1.3.20 setZero()

```
void StepperMC::setZero ( )
```

Set the current Position as zero. Use only on standstill.

Definition at line 308 of file StepperMC.cpp.

#### 4.1.3.21 stop()

```
void StepperMC::stop ( )
```

Set new target position for fast stop from current postion.

Definition at line 286 of file StepperMC.cpp.

The documentation for this class was generated from the following files:

- StepperMC.h
- StepperMC.cpp

# Chapter 5

# File Documentation

## 5.1 StepperMC.h

```
#ifndef StepperMC_h
#define StepperMC_h
#include <Arduino.h>

class StepperMC
{
public:
  StepperMC(uint8_t pin1, uint8_t pin2, uint8_t pin3, uint8_t pin4, uint16_t steps = 4096);

  StepperMC(uint8_t pinDir, uint8_t pinStep, uint16_t steps = 4096);

  void handle();

  void setIncrements(int32_t pos);

  void setIncrementsRelative(int32_t steps);

  int32_t getIncrements();

  void setPosition(float pos);

  void setPositionRelative(float pos);

  float getPosition();

  void moveTarget();

  bool inTarget();

  void stop();

  void setZero();

  void setSpeed(uint16_t freqMax, uint16_t acc = 0);

  void adjustZero(int32_t steps);

  void setBacklash(int32_t steps);

  void setGearRatio(int32_t motor, int32_t load);

  void setFeedConst(float feed);

  void setModulo(uint16_t steps = 0);

  void setUnlimited();

  void setPositionLimit(float lower, float upper);

  void reverseDir(bool neg);

  void setPowersaveTime(uint16_t seconds);

private:
  void _init(uint16_t steps);
  void _calcDelay();
  int32_t _trimModulo(int32_t pos);
  int32_t _diffModulo(int32_t diff);
  bool _stepUp();
  bool _stepDown();
  void _step();
  void _powerOff();
  enum {
    stp4Wire,
    stp2Wire
  } _interface;
  bool _isModulo;
  bool _isLimited;
  bool _negDir;
  uint16_t _stepsTurn;
  int32_t _stepAct;
  int32_t _stepTarget;
  int32_t _backlash;
  int32_t _backlashAct;
  int32_t _stepMotor;
  int32_t _stepsModulo;
  int32_t _upperLimit;
  int32_t _lowerLimit;
  float _feedConst;
  float _gearRatio;
  // motor pin numbers
  uint8_t _pin1;
  uint8_t _pin2;
  uint8_t _pin3;
  uint8_t _pin4;
  // timing
  unsigned long _delayPowersave;
  unsigned long _timeLastStep;
  unsigned long _delayStep;
  enum {
    dirStop,
    dirPos,
    dirNeg
  } _direction;
  // ramp
  float _cycle;
  float _cycleMin;
  float _cycleMax;
  float _rampConst;
  int32_t _rampStep;
  int32_t _stepsStop;
};

#endif
```

## 5.2 StepperMC.cpp

```
#include <Arduino.h>
#include "StepperMC.h"

// stepping scheme for the motor
const uint8_t phase_scheme[8][4] =
{
  {1,1,0,0},
  {0,1,0,0},
  {0,1,1,0},
  {0,0,1,0},
  {0,0,1,1},
  {0,0,0,1},
  {1,0,0,1},
  {1,0,0,0}
};

// constructor
StepperMC::StepperMC(uint8_t pin1, uint8_t pin2, uint8_t pin3, uint8_t pin4, uint16_t steps)
{
  // Initialize variables
  _init(steps);
  // Set Stepper interface
  _interface = stp4Wire;
  // Arduino pins for the motor control connection:
  _pin1 = pin1;
  _pin2 = pin2;
  _pin3 = pin3;
  _pin4 = pin4;
  // setup the pins on the microcontroller:
  pinMode(_pin1, OUTPUT);
  pinMode(_pin2, OUTPUT);
  pinMode(_pin3, OUTPUT);
  pinMode(_pin4, OUTPUT);
  // and start in idle mode
  _powerOff();
}

// constructor
StepperMC::StepperMC(uint8_t pinDir, uint8_t pinStep, uint16_t steps)
{
  // Initialize variables
  _init(steps);
  // Set Stepper interface
  _interface = stp2Wire;
  // Arduino pins for the motor control connection:
  _pin1 = pinDir;
  _pin2 = pinStep;
  _pin3 = 255;
  _pin4 = 255;
  // setup the pins on the microcontroller:
  pinMode(_pin1, OUTPUT);
  pinMode(_pin2, OUTPUT);
}

void StepperMC::_init(uint16_t steps)
{
  _stepAct = 0;
  _stepTarget = 0;
  _direction = dirStop;
  _backlash = 0;
  _backlashAct = 0;
  _stepMotor = 0;
  _isModulo = false;
  _isLimited = false;
  _stepsTurn = steps;
  _stepsModulo = 0;
  _feedConst = _stepsTurn / 360.0;
  _gearRatio = 1.0;
  _upperLimit = 0x7fffffff;
  _lowerLimit = 0x80000001;
  _delayStep = 1250;
  _cycle = 0;
  _cycleMin = 0;
  _cycleMax = 0;
  _rampConst = 0;
  _rampStep = 0;
  _stepsStop = 0;
  _delayPowersave = 1000000;
  _timeLastStep = micros() + _delayStep;
}

// cyclic handle of motion (call in loop)
void StepperMC::handle()
{
  // check if next step can be executed
  unsigned long now = micros();
  if (now > _timeLastStep + _delayStep)
  {
    // get new direction and step delay
    _calcDelay();
    // do one step in the right direction
    if (_direction == dirPos)
    {
      // count step only when backlash fully compensated
      if (_stepUp())
      {
        _stepAct = _trimModulo(_stepAct + 1);
      }
      _timeLastStep = now;
    }
    else if (_direction == dirNeg)
    {
      // count step only when backlash fully compensated
      if(_stepDown())
      {
        _stepAct = _trimModulo(_stepAct - 1);
      }
      _timeLastStep = now;
    }
    // activate powersave on standstill
    if ((_delayPowersave > 0) && (now > _timeLastStep + _delayPowersave))
    {
      _powerOff();
    }
  }
}

// update ramp and calculate new step delay
void StepperMC::_calcDelay()
{
  // get distance to target
  int32_t diff = _diffModulo(_stepTarget - _stepAct);
  // no ramp?
  if (_rampConst == 0)
  {
    // set direction and exit
    _direction = (diff > 0) ? dirPos : (diff < 0) ? dirNeg : dirStop;
    return;
  }
  // Stop when in Target
  if ((diff == 0) && (_stepsStop <= 5))
  {
    _direction = dirStop;
    _cycle = _cycleMax;
    _delayStep = 0;
    _rampStep = 0;
    return;
  }
  // detect necessary switch between acceleration and deceleration
  if (diff > 0) // positive turn needed?
  {
    if (_rampStep > 0) // accelerating or constant speed?
    {
      if ((_stepsStop >= diff) || (_direction == dirNeg))
      {
        // start deceleration
        _rampStep = -_stepsStop;
      }
    }
    else if (_rampStep < 0) // decelerating?
    {
      if ((_stepsStop < diff) && (_direction == dirPos))
      {
        // accelerate again
        _rampStep = -_rampStep;
      }
    }
  }
  else if (diff < 0) // negative turn needed?
  {
    if (_rampStep > 0) // accelerating or constant speed?
    {
      if ((_stepsStop >= -diff) || (_direction == dirPos))
      {
        // start deceleration
        _rampStep = -_stepsStop;
      }
    }
    else if (_rampStep < 0) // decelerating?
    {
      if ((_stepsStop < -diff) && (_direction == dirNeg))
      {
        // accelerate again
        _rampStep = -_rampStep;
      }
    }
  }
  // on zero crossing
  if (_rampStep == 0)
  {
    // set required direction to target and reinitialize cycle time
    _direction = (diff > 0) ? dirPos : dirNeg;
    _cycle = _cycleMax;
    // get new stopping distance
    _stepsStop = _rampConst / (_cycle * _cycle);
    // next rampStep
    _rampStep++;
  }
  else
  {
    // update cycle time when not final speed reached
    if (_cycle > _cycleMin || _rampStep < 0)
    {
      _cycle = _cycle - ((2.0 * _cycle) / ((4 * _rampStep) + 1));
      // get new stopping distance
      _stepsStop = _rampConst / (_cycle * _cycle);
      // next rampStep
      _rampStep++;
    }
  }
  // set current delay
  _delayStep = (unsigned long)_cycle;
}

// set new target position
void StepperMC::setIncrements(int32_t pos)
{
  if (_isLimited)
  {
    pos = min(max(pos, _lowerLimit), _upperLimit);
  }
  // enforce modulo
  _stepTarget = _trimModulo(pos);
}

// set relative target position
void StepperMC::setIncrementsRelative(int32_t steps)
{
  setIncrements(_stepTarget + steps);
}

// set new target position
void StepperMC::setPosition(float pos)
{
  setIncrements((int32_t)(pos * _feedConst));
}

// set new target position relative
void StepperMC::setPositionRelative(float pos)
{
  setIncrementsRelative((int32_t)(pos * _feedConst));
}

// automatic trim position in modulo range
int32_t StepperMC::_trimModulo(int32_t pos)
{
  if (_isModulo)
  {
    if (pos >= _stepsModulo)
    {
      pos -= _stepsModulo;
    }
    if (pos < 0)
    {
      pos += _stepsModulo;
    }
  }
  return pos;
}

// automatic trim position difference in modulo range
int32_t StepperMC::_diffModulo(int32_t diff)
{
  if (_isModulo)
  {
    if (diff > (_stepsModulo » 1))
    {
      diff -= _stepsModulo;
    }
    if (diff < -(_stepsModulo » 1))
    {
      diff += _stepsModulo;
    }
  }
  return diff;
}

// return actual position
int32_t StepperMC::getIncrements()
{
  return (_stepAct);
}

// get new cuurrent position
float StepperMC::getPosition()
{
  return (float) _stepAct / _feedConst;
}

// check if target position reached
bool StepperMC::inTarget()
{
  return (_stepTarget == _stepAct);
}

void StepperMC::stop()
{
  if (_direction == dirPos)
  {
    _stepTarget = _trimModulo(_stepAct + _stepsStop);
  }
  else if (_direction == dirNeg)
  {
    _stepTarget = _trimModulo(_stepAct - _stepsStop);
  }
```

```
}

// wait and handle steps until target position reached
void StepperMC::moveTarget()
{
  while (!inTarget())
  {
    handle();
  }
}

// set actual and target position to zero
void StepperMC::setZero()
{
  _stepAct = 0;
  _stepTarget = 0;
}

// adjust zero position by some steps
void StepperMC::adjustZero(int32_t steps)
{
  _stepAct -= steps;
}

// set backlash compensation
void StepperMC::setBacklash(int32_t steps)
{
  _backlash = steps;
}

// set dynamic speed ramping, acc = 0 means constant speed
// do not use during active movement
void StepperMC::setSpeed(uint16_t freq, uint16_t acc)
{
  if (freq > 0)
  {
    if (acc == 0)
    {
      _delayStep = 1000000UL / freq;
      _rampConst = 0;
    }
    else
    {
      _cycleMin = 1e6 / (float)freq;
      _cycleMax = 676e3 * sqrt(2.0 / ((float)acc));
      _cycle = _cycleMax;
      _rampConst = (5e11 / (float)acc);
    }
  }
}

// set gear ratio (set before setting feed constant and modulo!)
void StepperMC::setGearRatio(int32_t teethMotor, int32_t teethLoad)
{
  _gearRatio = (float)teethMotor / (float)teethLoad;
}

// set feedrate per load turn (default 360)
void StepperMC::setFeedConst(float feed)
{
  _feedConst = _stepsTurn / (feed * _gearRatio);
}

// make this a modulo axis.
// default (steps = 0): modulo distance = one turn on load side (nothing else is sensible)
// take care with uneven gear ratios! no handling of fractional parts!
void StepperMC::setModulo(uint16_t steps)
{
  _isModulo = true;
  _isLimited = false;
  if (steps == 0)
  {
    _stepsModulo = _stepsTurn / _gearRatio;
  }
  else
  {
    _stepsModulo = steps;
  }
}

// make unlimited. remove all limits and modulo
void StepperMC::setUnlimited()
{
  _isLimited = false;
  _isModulo = false;
  _lowerLimit = 0x80000001;
  _upperLimit = 0x7fffffff;
  _stepsModulo = 0;
}

// set software limits
void StepperMC::setPositionLimit(float lower, float upper)
{
  _isLimited = true;
  _isModulo = false;
  _lowerLimit = lower * _feedConst;
  _upperLimit = upper * _feedConst;
}

// invert direction
void StepperMC::reverseDir(bool neg)
{
  _negDir = neg;
}

void StepperMC::setPowersaveTime(uint16_t seconds)
{
  _delayPowersave = 1000000UL * seconds;
}

bool StepperMC::_stepUp()
{
  _stepMotor++;
  _step();
  if (_backlashAct < _backlash - 1)
  {
    _backlashAct++;
    return false;
  }
  return true;
}

bool StepperMC::_stepDown()
{
  _stepMotor--;
  _step();
  if (_backlashAct > 0)
  {
    _backlashAct--;
    return false;
  }
  return true;
}

// execute one step
void StepperMC::_step()
{
  if (_interface == stp4Wire)
  {
    int phase = (int)(_stepMotor & 0x07);
    if (_negDir)
    {
      // invert direction
      phase = 7 - phase;
    }
    digitalWrite(_pin1, phase_scheme[phase][0]);
    digitalWrite(_pin2, phase_scheme[phase][1]);
    digitalWrite(_pin3, phase_scheme[phase][2]);
    digitalWrite(_pin4, phase_scheme[phase][3]);
  }
  else if (_interface == stp2Wire)
  {
    digitalWrite(_pin1, (_direction == dirPos) != _negDir);
    digitalWrite(_pin2, true);
    delayMicroseconds(1);
    digitalWrite(_pin2, false);
  }
}

// switch power off
void StepperMC::_powerOff()
{
  // powerOff is sensible only with 4 wire interface
  if (_interface == stp4Wire)
  {
    digitalWrite(_pin1, 0);
    digitalWrite(_pin2, 0);
    digitalWrite(_pin3, 0);
    digitalWrite(_pin4, 0);
  }
}
```

# Index