# 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

```
00001 #ifndef StepperMC_h
00002 #define StepperMC_h
00003 #include <Arduino.h>
00004
00005 class StepperMC
00006 {
00007 public:
00014   StepperMC(uint8_t pin1, uint8_t pin2, uint8_t pin3, uint8_t pin4, uint16_t steps = 4096);
00015
00020   StepperMC(uint8_t pinDir, uint8_t pinStep, uint16_t steps = 4096);
00021
00023   void handle();
00024
00027   void setIncrements(int32_t pos);
00028
00031   void setIncrementsRelative(int32_t steps);
00032
00035   int32_t getIncrements();
00036
00039   void setPosition(float pos);
00040
00043   void setPositionRelative(float pos);
00044
00047   float getPosition();
00048
00050   void moveTarget();
00051
00054   bool inTarget();
00055
00057   void stop();
00058
00060   void setZero();
00061
00066   void setSpeed(uint16_t freqMax, uint16_t acc = 0);
00067
00070   void adjustZero(int32_t steps);
00071
00074   void setBacklash(int32_t steps);
00075
00079   void setGearRatio(int32_t motor, int32_t load);
00080
00083   void setFeedConst(float feed);
00084
00088   void setModulo(uint16_t steps = 0);
00089
00091   void setUnlimited();
00092
00097   void setPositionLimit(float lower, float upper);
00098
00101   void reverseDir(bool neg);
00102
00105   void setPowersaveTime(uint16_t seconds);
00106
00107 private:
00108   void _init(uint16_t steps);
00109   void _calcDelay();
00110   int32_t _trimModulo(int32_t pos);
00111   int32_t _diffModulo(int32_t diff);
```

```
00112   bool _stepUp();
00113   bool _stepDown();
00114   void _step();
00115   void _powerOff();
00116   enum {
00117     stp4Wire,
00118     stp2Wire
00119   } _interface;
00120   bool _isModulo;
00121   bool _isLimited;
00122   bool _negDir;
00123   uint16_t _stepsTurn;
00124   int32_t _stepAct;
00125   int32_t _stepTarget;
00126   int32_t _backlash;
00127   int32_t _backlashAct;
00128   int32_t _stepMotor;
00129   int32_t _stepsModulo;
00130   int32_t _upperLimit;
00131   int32_t _lowerLimit;
00132   float _feedConst;
00133   float _gearRatio;
00134   // motor pin numbers
00135   uint8_t _pin1;
00136   uint8_t _pin2;
00137   uint8_t _pin3;
00138   uint8_t _pin4;
00139   // timing
00140   unsigned long _delayPowersave;
00141   unsigned long _timeLastStep;
00142   unsigned long _delayStep;
00143   enum {
00144     dirStop,
00145     dirPos,
00146     dirNeg
00147   } _direction;
00148   // ramp
00149   float _cycle;
00150   float _cycleMin;
00151   float _cycleMax;
00152   float _rampConst;
00153   int32_t _rampStep;
00154   int32_t _stepsStop;
00155 };
00156
00157 #endif
```

## 5.2 StepperMC.cpp

```
00001 #include <Arduino.h>
00002 #include "StepperMC.h"
00003
00004 // stepping scheme for the motor
00005 const uint8_t phase_scheme[8][4] =
00006 {
00007   {1,1,0,0},
00008   {0,1,0,0},
00009   {0,1,1,0},
00010   {0,0,1,0},
00011   {0,0,1,1},
00012   {0,0,0,1},
00013   {1,0,0,1},
00014   {1,0,0,0}
00015 };
00016
00017 // constructor
00018 StepperMC::StepperMC(uint8_t pin1, uint8_t pin2, uint8_t pin3, uint8_t pin4, uint16_t steps)
00019 {
00020   // Initialize variables
00021   _init(steps);
00022   // Set Stepper interface
00023   _interface = stp4Wire;
00024   // Arduino pins for the motor control connection:
00025   _pin1 = pin1;
00026   _pin2 = pin2;
00027   _pin3 = pin3;
00028   _pin4 = pin4;
00029   // setup the pins on the microcontroller:
00030   pinMode(_pin1, OUTPUT);
00031   pinMode(_pin2, OUTPUT);
00032   pinMode(_pin3, OUTPUT);
00033   pinMode(_pin4, OUTPUT);
00034   // and start in idle mode
```

```
00035   _powerOff();
00036 }
00037
00038 // constructor
00039 StepperMC::StepperMC(uint8_t pinDir, uint8_t pinStep, uint16_t steps)
00040 {
00041   // Initialize variables
00042   _init(steps);
00043   // Set Stepper interface
00044   _interface = stp2Wire;
00045   // Arduino pins for the motor control connection:
00046   _pin1 = pinDir;
00047   _pin2 = pinStep;
00048   _pin3 = 255;
00049   _pin4 = 255;
00050   // setup the pins on the microcontroller:
00051   pinMode(_pin1, OUTPUT);
00052   pinMode(_pin2, OUTPUT);
00053 }
00054
00055 void StepperMC::_init(uint16_t steps)
00056 {
00057   _stepAct = 0;
00058   _stepTarget = 0;
00059   _direction = dirStop;
00060   _backlash = 0;
00061   _backlashAct = 0;
00062   _stepMotor = 0;
00063   _isModulo = false;
00064   _isLimited = false;
00065   _stepsTurn = steps;
00066   _stepsModulo = 0;
00067   _feedConst = _stepsTurn / 360.0;
00068   _gearRatio = 1.0;
00069   _upperLimit = 0x7fffffff;
00070   _lowerLimit = 0x80000001;
00071   _delayStep = 1250;
00072   _cycle = 0;
00073   _cycleMin = 0;
00074   _cycleMax = 0;
00075   _rampConst = 0;
00076   _rampStep = 0;
00077   _stepsStop = 0;
00078   _delayPowersave = 1000000;
00079   _timeLastStep = micros() + _delayStep;
00080 }
00081
00082 // cyclic handle of motion (call in loop)
00083 void StepperMC::handle()
00084 {
00085   // check if next step can be executed
00086   unsigned long now = micros();
00087   if (now > _timeLastStep + _delayStep)
00088   {
00089     // get new direction and step delay
00090     _calcDelay();
00091     // do one step in the right direction
00092     if (_direction == dirPos)
00093     {
00094       // count step only when backlash fully compensated
00095       if (_stepUp())
00096       {
00097         _stepAct = _trimModulo(_stepAct + 1);
00098       }
00099       _timeLastStep = now;
00100     }
00101     else if (_direction == dirNeg)
00102     {
00103       // count step only when backlash fully compensated
00104       if(_stepDown())
00105       {
00106         _stepAct = _trimModulo(_stepAct - 1);
00107       }
00108       _timeLastStep = now;
00109     }
00110     // activate powersave on standstill
00111     if ((_delayPowersave > 0) && (now > _timeLastStep + _delayPowersave))
00112     {
00113       _powerOff();
00114     }
00115   }
00116 }
00117
00118 // update ramp and calculate new step delay
00119 void StepperMC::_calcDelay()
00120 {
00121   // get distance to target
```

```
00122   int32_t diff = _diffModulo(_stepTarget - _stepAct);
00123   // no ramp?
00124   if (_rampConst == 0)
00125   {
00126     // set direction and exit
00127     _direction = (diff > 0) ? dirPos : (diff < 0) ? dirNeg : dirStop;
00128     return;
00129   }
00130   // Stop when in Target
00131   if ((diff == 0) && (_stepsStop <= 5))
00132   {
00133     _direction = dirStop;
00134     _cycle = _cycleMax;
00135     _delayStep = 0;
00136     _rampStep = 0;
00137     return;
00138   }
00139   // detect necessary switch between acceleration and deceleration
00140   if (diff > 0) // positive turn needed?
00141   {
00142     if (_rampStep > 0) // accelerating or constant speed?
00143     {
00144       if ((_stepsStop >= diff) || (_direction == dirNeg))
00145       {
00146         // start deceleration
00147         _rampStep = -_stepsStop;
00148       }
00149     }
00150     else if (_rampStep < 0) // decelerating?
00151     {
00152       if ((_stepsStop < diff) && (_direction == dirPos))
00153       {
00154         // accelerate again
00155         _rampStep = -_rampStep;
00156       }
00157     }
00158   }
00159   else if (diff < 0) // negative turn needed?
00160   {
00161     if (_rampStep > 0) // accelerating or constant speed?
00162     {
00163       if ((_stepsStop >= -diff) || (_direction == dirPos))
00164       {
00165         // start deceleration
00166         _rampStep = -_stepsStop;
00167       }
00168     }
00169     else if (_rampStep < 0) // decelerating?
00170     {
00171       if ((_stepsStop < -diff) && (_direction == dirNeg))
00172       {
00173         // accelerate again
00174         _rampStep = -_rampStep;
00175       }
00176     }
00177   }
00178   // on zero crossing
00179   if (_rampStep == 0)
00180   {
00181     // set required direction to target and reinitialize cycle time
00182     _direction = (diff > 0) ? dirPos : dirNeg;
00183     _cycle = _cycleMax;
00184     // get new stopping distance
00185     _stepsStop = _rampConst / (_cycle * _cycle);
00186     // next rampStep
00187     _rampStep++;
00188   }
00189   else
00190   {
00191     // update cycle time when not final speed reached
00192     if (_cycle > _cycleMin || _rampStep < 0)
00193     {
00194       _cycle = _cycle - ((2.0 * _cycle) / ((4 * _rampStep) + 1));
00195       // get new stopping distance
00196       _stepsStop = _rampConst / (_cycle * _cycle);
00197       // next rampStep
00198       _rampStep++;
00199     }
00200   }
00201   // set current delay
00202   _delayStep = (unsigned long)_cycle;
00203 }
00204
00205 // set new target position
00206 void StepperMC::setIncrements(int32_t pos)
00207 {
00208   if (_isLimited)
```

```
00209   {
00210     pos = min(max(pos, _lowerLimit), _upperLimit);
00211   }
00212   // enforce modulo
00213   _stepTarget = _trimModulo(pos);
00214 }
00215
00216 // set relative target position
00217 void StepperMC::setIncrementsRelative(int32_t steps)
00218 {
00219   setIncrements(_stepTarget + steps);
00220 }
00221
00222 // set new target position
00223 void StepperMC::setPosition(float pos)
00224 {
00225   setIncrements((int32_t)(pos * _feedConst));
00226 }
00227
00228 // set new target position relative
00229 void StepperMC::setPositionRelative(float pos)
00230 {
00231   setIncrementsRelative((int32_t)(pos * _feedConst));
00232 }
00233
00234 // automatic trim position in modulo range
00235 int32_t StepperMC::_trimModulo(int32_t pos)
00236 {
00237   if (_isModulo)
00238   {
00239     if (pos >= _stepsModulo)
00240     {
00241       pos -= _stepsModulo;
00242     }
00243     if (pos < 0)
00244     {
00245       pos += _stepsModulo;
00246     }
00247   }
00248   return pos;
00249 }
00250
00251 // automatic trim position difference in modulo range
00252 int32_t StepperMC::_diffModulo(int32_t diff)
00253 {
00254   if (_isModulo)
00255   {
00256     if (diff > (_stepsModulo » 1))
00257     {
00258       diff -= _stepsModulo;
00259     }
00260     if (diff < -(_stepsModulo » 1))
00261     {
00262       diff += _stepsModulo;
00263     }
00264   }
00265   return diff;
00266 }
00267
00268 // return actual position
00269 int32_t StepperMC::getIncrements()
00270 {
00271   return (_stepAct);
00272 }
00273
00274 // get new cuurrent position
00275 float StepperMC::getPosition()
00276 {
00277   return (float) _stepAct / _feedConst;
00278 }
00279
00280 // check if target position reached
00281 bool StepperMC::inTarget()
00282 {
00283   return (_stepTarget == _stepAct);
00284 }
00285
00286 void StepperMC::stop()
00287 {
00288   if (_direction == dirPos)
00289   {
00290     _stepTarget = _trimModulo(_stepAct + _stepsStop);
00291   }
00292   else if (_direction == dirNeg)
00293   {
00294     _stepTarget = _trimModulo(_stepAct - _stepsStop);
00295   }
```

```
00296 }
00297
00298 // wait and handle steps until target position reached
00299 void StepperMC::moveTarget()
00300 {
00301   while (!inTarget())
00302   {
00303     handle();
00304   }
00305 }
00306
00307 // set actual and target position to zero
00308 void StepperMC::setZero()
00309 {
00310   _stepAct = 0;
00311   _stepTarget = 0;
00312 }
00313
00314 // adjust zero position by some steps
00315 void StepperMC::adjustZero(int32_t steps)
00316 {
00317   _stepAct -= steps;
00318 }
00319
00320 // set backlash compensation
00321 void StepperMC::setBacklash(int32_t steps)
00322 {
00323   _backlash = steps;
00324 }
00325
00326 // set dynamic speed ramping, acc = 0 means constant speed
00327 // do not use during active movement
00328 void StepperMC::setSpeed(uint16_t freq, uint16_t acc)
00329 {
00330   if (freq > 0)
00331   {
00332     if (acc == 0)
00333     {
00334       _delayStep = 1000000UL / freq;
00335       _rampConst = 0;
00336     }
00337     else
00338     {
00339       _cycleMin = 1e6 / (float)freq;
00340       _cycleMax = 676e3 * sqrt(2.0 / ((float)acc));
00341       _cycle = _cycleMax;
00342       _rampConst = (5e11 / (float)acc);
00343     }
00344   }
00345 }
00346
00347 // set gear ratio (set before setting feed constant and modulo!)
00348 void StepperMC::setGearRatio(int32_t teethMotor, int32_t teethLoad)
00349 {
00350   _gearRatio = (float)teethMotor / (float)teethLoad;
00351 }
00352
00353 // set feedrate per load turn (default 360)
00354 void StepperMC::setFeedConst(float feed)
00355 {
00356   _feedConst = _stepsTurn / (feed * _gearRatio);
00357 }
00358
00359 // make this a modulo axis.
00360 // default (steps = 0): modulo distance = one turn on load side (nothing else is sensible)
00361 // take care with uneven gear ratios! no handling of fractional parts!
00362 void StepperMC::setModulo(uint16_t steps)
00363 {
00364   _isModulo = true;
00365   _isLimited = false;
00366   if (steps == 0)
00367   {
00368     _stepsModulo = _stepsTurn / _gearRatio;
00369   }
00370   else
00371   {
00372     _stepsModulo = steps;
00373   }
00374 }
00375
00376 // make unlimited. remove all limits and modulo
00377 void StepperMC::setUnlimited()
00378 {
00379   _isLimited = false;
00380   _isModulo = false;
00381   _lowerLimit = 0x80000001;
00382   _upperLimit = 0x7fffffff;
```

```
00383   _stepsModulo = 0;
00384 }
00385
00386 // set software limits
00387 void StepperMC::setPositionLimit(float lower, float upper)
00388 {
00389   _isLimited = true;
00390   _isModulo = false;
00391   _lowerLimit = lower * _feedConst;
00392   _upperLimit = upper * _feedConst;
00393 }
00394
00395 // invert direction
00396 void StepperMC::reverseDir(bool neg)
00397 {
00398   _negDir = neg;
00399 }
00400
00401 void StepperMC::setPowersaveTime(uint16_t seconds)
00402 {
00403   _delayPowersave = 1000000UL * seconds;
00404 }
00405
00406 bool StepperMC::_stepUp()
00407 {
00408   _stepMotor++;
00409   _step();
00410   if (_backlashAct < _backlash - 1)
00411   {
00412     _backlashAct++;
00413     return false;
00414   }
00415   return true;
00416 }
00417
00418 bool StepperMC::_stepDown()
00419 {
00420   _stepMotor--;
00421   _step();
00422   if (_backlashAct > 0)
00423   {
00424     _backlashAct--;
00425     return false;
00426   }
00427   return true;
00428 }
00429
00430 // execute one step
00431 void StepperMC::_step()
00432 {
00433   if (_interface == stp4Wire)
00434   {
00435     int phase = (int)(_stepMotor & 0x07);
00436     if (_negDir)
00437     {
00438       // invert direction
00439       phase = 7 - phase;
00440     }
00441     digitalWrite(_pin1, phase_scheme[phase][0]);
00442     digitalWrite(_pin2, phase_scheme[phase][1]);
00443     digitalWrite(_pin3, phase_scheme[phase][2]);
00444     digitalWrite(_pin4, phase_scheme[phase][3]);
00445   }
00446   else if (_interface == stp2Wire)
00447   {
00448     digitalWrite(_pin1, (_direction == dirPos) != _negDir);
00449     digitalWrite(_pin2, true);
00450     delayMicroseconds(1);
00451     digitalWrite(_pin2, false);
00452   }
00453 }
00454
00455 // switch power off
00456 void StepperMC::_powerOff()
00457 {
00458   // powerOff is sensible only with 4 wire interface
00459   if (_interface == stp4Wire)
00460   {
00461     digitalWrite(_pin1, 0);
00462     digitalWrite(_pin2, 0);
00463     digitalWrite(_pin3, 0);
00464     digitalWrite(_pin4, 0);
00465   }
00466 }
```

# Index