AccelStepper Class Reference

Support for stepper motors with acceleration etc. More...

```
#include <AccelStepper.h>
```

Public Types

enum **MotorInterfaceType** {
  FUNCTION = 0 , **DRIVER** = 1 , **FULL2WIRE** = 2 , **FULL3WIRE** = 3 ,
  FULL4WIRE = 4 , **HALF3WIRE** = 6 , **HALF4WIRE** = 8
}
Symbolic names for number of pins. Use this in the pins argument the **AccelStepper** constructor to provide a symbolic name for the number of pins to use. More...

Public Member Functions

	AccelStepper (uint8_t interface=**AccelStepper::FULL4WIRE**, uint8_t pin1=2, uint8_t pin2=3, uint8_t pin3=4, uint8_t pin4=5, bool enable=true)
	AccelStepper (void(*forward)(), void(*backward)())
void	**moveTo** (long absolute)
void	**move** (long relative)
boolean	**run** ()
boolean	**runSpeed** ()
void	**setMaxSpeed** (float **speed**)
float	**maxSpeed** ()
void	**setAcceleration** (float **acceleration**)
float	**acceleration** ()
void	**setSpeed** (float **speed**)
float	**speed** ()
long	**distanceToGo** ()
long	**targetPosition** ()
long	**currentPosition** ()
void	**setCurrentPosition** (long position)
void	**runToPosition** ()
boolean	**runSpeedToPosition** ()
void	**runToNewPosition** (long position)
void	**stop** ()
virtual void	**disableOutputs** ()
virtual void	**enableOutputs** ()
void	**setMinPulseWidth** (unsigned int minWidth)
void	**setEnablePin** (uint8_t enablePin=0xff)
void	**setPinsInverted** (bool directionInvert=false, bool stepInvert=false, bool enableInvert=false)

void	**setPinsInverted** (bool pin1Invert, bool pin2Invert, bool pin3Invert, bool pin4Invert, bool enableInvert)
bool	**isRunning** ()
virtual	**~AccelStepper** () Virtual destructor to prevent warnings during delete.

Protected Types

enum	**Direction** { **DIRECTION_CCW** = 0 , **DIRECTION_CW** = 1 } Direction indicator Symbolic names for the direction the motor is turning. More...

Protected Member Functions

virtual unsigned long	**computeNewSpeed** ()
virtual void	**setOutputPins** (uint8_t mask)
virtual void	**step** (long step)
long	**stepForward** ()
long	**stepBackward** ()
virtual void	**step0** (long **step**)
virtual void	**step1** (long **step**)
virtual void	**step2** (long **step**)
virtual void	**step3** (long **step**)
virtual void	**step4** (long **step**)
virtual void	**step6** (long **step**)
virtual void	**step8** (long **step**)

Protected Attributes

boolean	**_direction**
unsigned long	**_stepInterval**

Detailed Description

Support for stepper motors with acceleration etc.

This defines a single 2 or 4 pin stepper motor, or stepper moter with fdriver chip, with optional acceleration, deceleration, absolute positioning commands etc. Multiple simultaneous steppers are supported, all moving at different speeds and accelerations.

Operation

This module operates by computing a step time in microseconds. The step time is recomputed after each step and after speed and acceleration parameters are changed by the caller. The time of each step is recorded in microseconds. The **run()** function steps the motor once if a new step is due. The **run()** function must be called frequently until the motor is in the desired position, after which time **run()** will do nothing.

Positioning

Positions are specified by a signed long integer. At construction time, the current position of the motor is consider to be 0. Positive positions are clockwise from the initial position; negative positions are anticlockwise. The current position can be altered for instance after initialization positioning.

Caveats

This is an open loop controller: If the motor stalls or is oversped, **AccelStepper** will not have a correct idea of where the motor really is (since there is no feedback of the motor's real position. We only know where we *think* it is, relative to the initial starting point).

Performance

The fastest motor speed that can be reliably supported is about 4000 steps per second at a clock frequency of 16 MHz on Arduino such as Uno etc. Faster processors can support faster stepping speeds. However, any speed less than that down to very slow speeds (much less than one per second) are also supported, provided the **run()** function is called frequently enough to step the motor whenever required for the speed set. Calling **setAcceleration()** is expensive, since it requires a square root to be calculated.

Gregor Christandl reports that with an Arduino Due and a simple test program, he measured 43163 steps per second using **runSpeed()**, and 16214 steps per second using **run()**;

Member Enumeration Documentation

◆ Direction

enum **AccelStepper::Direction** protected

Direction indicator Symbolic names for the direction the motor is turning.

Enumerator	
DIRECTION_CCW	Counter-Clockwise.
DIRECTION_CW	Clockwise.

◆ MotorInterfaceType

enum **AccelStepper::MotorInterfaceType**

Symbolic names for number of pins. Use this in the pins argument the **AccelStepper** constructor to provide a symbolic name for the number of pins to use.

Enumerator	
FUNCTION	Use the functional interface, implementing your own driver functions (internal use only)
DRIVER	Stepper Driver, 2 driver pins required.
FULL2WIRE	2 wire stepper, 2 motor pins required
FULL3WIRE	3 wire stepper, such as HDD spindle, 3 motor pins required
FULL4WIRE	4 wire full stepper, 4 motor pins required
HALF3WIRE	3 wire half stepper, such as HDD spindle, 3 motor pins required
HALF4WIRE	4 wire half stepper, 4 motor pins required

Constructor & Destructor Documentation

◆ AccelStepper() [1/2]

```
AccelStepper::AccelStepper ( uint8_t  interface = AccelStepper::FULL4WIRE,
                             uint8_t  pin1 = 2,
                             uint8_t  pin2 = 3,
                             uint8_t  pin3 = 4,
                             uint8_t  pin4 = 5,
                             bool     enable = true
                           )
```

Constructor. You can have multiple simultaneous steppers, all moving at different speeds and accelerations, provided you call their **run()** functions at frequent enough intervals. Current Position is set to 0, target position is set to 0. MaxSpeed and Acceleration default to 1.0. The motor pins will be initialised to OUTPUT mode during the constructor by a call to **enableOutputs()**.

Parameters

[in]	**interface**	Number of pins to interface to. Integer values are supported, but it is preferred to use the **MotorInterfaceType** symbolic names. **AccelStepper::DRIVER** (1) means a stepper driver (with Step and Direction pins). If an enable line is also needed, call **setEnablePin()** after construction. You may also invert the pins using **setPinsInverted()**. Caution: DRIVER implements a blocking delay of minPulseWidth microseconds (default 1us) for each step. You can change this with **setMinPulseWidth()**. **AccelStepper::FULL2WIRE** (2) means a 2 wire stepper (2 pins required). **AccelStepper::FULL3WIRE** (3) means a 3 wire stepper, such as HDD spindle (3 pins required). **AccelStepper::FULL4WIRE** (4) means a 4 wire stepper (4 pins required). **AccelStepper::HALF3WIRE** (6) means a 3 wire half stepper, such as HDD spindle (3 pins required) **AccelStepper::HALF4WIRE** (8) means a 4 wire half stepper (4 pins required) Defaults to **AccelStepper::FULL4WIRE** (4) pins.
[in]	**pin1**	Arduino digital pin number for motor pin 1. Defaults to pin 2. For a **AccelStepper::DRIVER** (interface==1), this is the Step input to the driver. Low to high transition means to step)
[in]	**pin2**	Arduino digital pin number for motor pin 2. Defaults to pin 3. For a **AccelStepper::DRIVER** (interface==1), this is the Direction input the driver. High means forward.
[in]	**pin3**	Arduino digital pin number for motor pin 3. Defaults to pin 4.
[in]	**pin4**	Arduino digital pin number for motor pin 4. Defaults to pin 5.
[in]	**enable**	If this is true (the default), **enableOutputs()** will be called to enable the output pins at construction time.

References **_direction**, **_stepInterval**, **DIRECTION_CCW**, **enableOutputs()**, **setAcceleration()**, and **setMaxSpeed()**.

◆ AccelStepper() [2/2]

```
AccelStepper::AccelStepper ( void(*)()  forward,
                             void(*)()  backward
                           )
```

Alternate Constructor which will call your own functions for forward and backward steps. You can have multiple simultaneous steppers, all moving at different speeds and accelerations, provided you call their **run()** functions at frequent enough intervals. Current Position is set to 0, target position is set to 0. MaxSpeed and Acceleration default to 1.0. Any motor initialization should happen before hand, no pins are used or initialized.

Parameters

`[in]`	**forward**	void-returning procedure that will make a forward step
`[in]`	**backward**	void-returning procedure that will make a backward step

References **_direction**, **_stepInterval**, **DIRECTION_CCW**, **setAcceleration()**, and **setMaxSpeed()**.

Member Function Documentation

◆ acceleration()

```
float AccelStepper::acceleration ( )
```

Returns the acceleration/deceleration rate configured for this stepper that was previously set by **setAcceleration()**;

Returns

The currently configured acceleration/deceleration

Referenced by **setAcceleration()**.

◆ computeNewSpeed()

unsigned long AccelStepper::computeNewSpeed ( ) protected virtual

Forces the library to compute a new instantaneous speed and set that as the current speed. It is called by the library:

- after each step
- after change to maxSpeed through **setMaxSpeed()**
- after change to acceleration through **setAcceleration()**
- after change to target position (relative or absolute) through **move()** or **moveTo()**

Returns
the new step interval

References **_direction**, **_stepInterval**, **DIRECTION_CCW**, **DIRECTION_CW**, and **distanceToGo()**.

Referenced by **moveTo()**, **run()**, **setAcceleration()**, and **setMaxSpeed()**.

◆ currentPosition()

long AccelStepper::currentPosition ( )

The current motor position.

Returns
the current motor position in steps. Positive is clockwise from the 0 position.

Referenced by **MultiStepper::moveTo()**.

◆ disableOutputs()

void AccelStepper::disableOutputs ( ) virtual

Disable motor pin outputs by setting them all LOW Depending on the design of your electronics this may turn off the power to the motor coils, saving power. This is useful to support Arduino low power modes: disable the outputs during sleep and then reenable with **enableOutputs()** before stepping again. If the enable Pin is defined, sets it to OUTPUT mode and clears the pin to disabled.

References **setOutputPins()**.

◆ distanceToGo()

long AccelStepper::distanceToGo ( )

The distance from the current position to the target position.

Returns
the distance from the current position to the target position in steps. Positive is clockwise from the current position.

Referenced by **computeNewSpeed()**, and **run()**.

◆ enableOutputs()

void AccelStepper::enableOutputs ( ) virtual

Enable motor pin outputs by setting the motor pins to OUTPUT mode. Called automatically by the constructor. If the enable Pin is defined, sets it to OUTPUT mode and sets the pin to enabled.

References **FULL3WIRE**, **FULL4WIRE**, **HALF3WIRE**, and **HALF4WIRE**.

Referenced by **AccelStepper()**.

◆ isRunning()

bool AccelStepper::isRunning ( )

Checks to see if the motor is currently running to a target

Returns
true if the speed is not zero or not at the target position

◆ maxSpeed()

float AccelStepper::maxSpeed ( )

Returns the maximum speed configured for this stepper that was previously set by **setMaxSpeed()**;

Returns
The currently configured maximum speed

Referenced by **MultiStepper::moveTo()**.

◆ move()

void AccelStepper::move (long relative)

Set the target position relative to the current position.

Parameters

[in] **relative** The desired position relative to the current position. Negative is anticlockwise from the current position.

References **moveTo()**.

Referenced by **stop()**.

◆ moveTo()

void AccelStepper::moveTo (long absolute)

Set the target position. The **run()** function will try to move the motor (at most one step per call) from the current position to the target position set by the most recent call to this function. Caution: **moveTo()** also recalculates the speed for the next step. If you are trying to use constant speed movements, you should call **setSpeed()** after calling **moveTo()**.

Parameters

[in] **absolute** The desired absolute position. Negative is anticlockwise from the 0 position.

References **computeNewSpeed()**.

Referenced by **move()**, **MultiStepper::moveTo()**, and **runToNewPosition()**.

◆ run()

boolean AccelStepper::run ( )

Poll the motor and step it if a step is due, implementing accelerations and decelerations to achieve the target position. You must call this as frequently as possible, but at least once per minimum step time interval, preferably in your main loop. Note that each call to **run()** will make at most one step, and then only when a step is due, based on the current speed and the time since the last step.

Returns

true if the motor is still running to the target position.

References **computeNewSpeed()**, **distanceToGo()**, and **runSpeed()**.

Referenced by **runToPosition()**.

◆ runSpeed()

boolean AccelStepper::runSpeed ( )

Poll the motor and step it if a step is due, implementing a constant speed as set by the most recent call to **setSpeed()**. You must call this as frequently as possible, but at least once per step interval,

Returns

true if the motor was stepped.

References **_direction**, **_stepInterval**, **DIRECTION_CW**, and **step()**.

Referenced by **run()**, **MultiStepper::run()**, and **runSpeedToPosition()**.

◆ runSpeedToPosition()

boolean AccelStepper::runSpeedToPosition ( )

Executes **runSpeed()** unless the targetPosition is reached. This function needs to be called often just like **runSpeed()** or **run()**. Will step the motor if a step is required at the currently selected speed unless the target position has been reached. Does not implement accelerations.

Returns

true if it stepped

References **_direction**, **DIRECTION_CCW**, **DIRECTION_CW**, and **runSpeed()**.

◆ runToNewPosition()

void AccelStepper::runToNewPosition (long position)

Moves the motor (with acceleration/deceleration) to the new target position and blocks until it is at position. Dont use this in event loops, since it blocks.

Parameters

`[in]` **position** The new target position.

References **moveTo()**, and **runToPosition()**.

◆ runToPosition()

void AccelStepper::runToPosition ( )

Moves the motor (with acceleration/deceleration) to the target position and blocks until it is at position. Dont use this in event loops, since it blocks.

References **run()**.

Referenced by **runToNewPosition()**.

◆ setAcceleration()

void AccelStepper::setAcceleration (float acceleration)

Sets the acceleration/deceleration rate.

Parameters

`[in]`	**acceleration**	The desired acceleration in steps per second per second. Must be > 0.0. This is an expensive call since it requires a square root to be calculated. Dont call more often than needed

References **acceleration()**, and **computeNewSpeed()**.

Referenced by **AccelStepper()**.

◆ setCurrentPosition()

void AccelStepper::setCurrentPosition (long position)

Resets the current position of the motor, so that wherever the motor happens to be right now is considered to be the new 0 position. Useful for setting a zero position on a stepper after an initial hardware positioning move. Has the side effect of setting the current motor speed to 0.

Parameters

`[in]`	**position**	The position in steps of wherever the motor happens to be right now.

References **_stepInterval**.

Referenced by **MultiStepper::run()**.

◆ setEnablePin()

void AccelStepper::setEnablePin (uint8_t enablePin = `0xff`)

Sets the enable pin number for stepper drivers. 0xFF indicates unused (default). Otherwise, if a pin is set, the pin will be turned on when **enableOutputs()** is called and switched off when **disableOutputs()** is called.

Parameters

`[in]` **enablePin** Arduino digital pin number for motor enable

See also

setPinsInverted

◆ setMaxSpeed()

void AccelStepper::setMaxSpeed (float speed)

Sets the maximum permitted speed. The **run()** function will accelerate up to the speed set by this function. Caution: the maximum speed achievable depends on your processor and clock speed. The default maxSpeed is 1.0 steps per second.

Parameters

`[in]` **speed** The desired maximum speed in steps per second. Must be > 0. Caution: Speeds that exceed the maximum speed supported by the processor may Result in non-linear accelerations and decelerations.

References **computeNewSpeed()**, and **speed()**.

Referenced by **AccelStepper()**.

◆ setMinPulseWidth()

void AccelStepper::setMinPulseWidth (unsigned int minWidth)

Sets the minimum pulse width allowed by the stepper driver. The minimum practical pulse width is approximately 20 microseconds. Times less than 20 microseconds will usually result in 20 microseconds or so.

Parameters

`[in]` **minWidth** The minimum pulse width in microseconds.

◆ setOutputPins()

void AccelStepper::setOutputPins (uint8_t mask) protected virtual

Low level function to set the motor output pins bit 0 of the mask corresponds to _pin[0] bit 1 of the mask corresponds to _pin[1] You can override this to impment, for example serial chip output insted of using the output pins directly

References **FULL3WIRE**, **FULL4WIRE**, **HALF3WIRE**, and **HALF4WIRE**.

Referenced by **disableOutputs()**, **step1()**, **step2()**, **step3()**, **step4()**, **step6()**, and **step8()**.

◆ setPinsInverted() [1/2]

```
void AccelStepper::setPinsInverted ( bool directionInvert = false,
                                     bool stepInvert = false,
                                     bool enableInvert = false
                                   )
```

Sets the inversion for stepper driver pins

Parameters

[in]	**directionInvert**	True for inverted direction pin, false for non-inverted
[in]	**stepInvert**	True for inverted step pin, false for non-inverted
[in]	**enableInvert**	True for inverted enable pin, false (default) for non-inverted

◆ setPinsInverted() [2/2]

```
void AccelStepper::setPinsInverted ( bool pin1Invert,
                                     bool pin2Invert,
                                     bool pin3Invert,
                                     bool pin4Invert,
                                     bool enableInvert
                                   )
```

Sets the inversion for 2, 3 and 4 wire stepper pins

Parameters

[in]	**pin1Invert**	True for inverted pin1, false for non-inverted
[in]	**pin2Invert**	True for inverted pin2, false for non-inverted
[in]	**pin3Invert**	True for inverted pin3, false for non-inverted
[in]	**pin4Invert**	True for inverted pin4, false for non-inverted
[in]	**enableInvert**	True for inverted enable pin, false (default) for non-inverted

◆ setSpeed()

void AccelStepper::setSpeed (float speed)

Sets the desired constant speed for use with **runSpeed()**.

Parameters

[in] **speed** The desired constant speed in steps per second. Positive is clockwise. Speeds of more than 1000 steps per second are unreliable. Very slow speeds may be set (eg 0.00027777 for once per hour, approximately. Speed accuracy depends on the Arduino crystal. Jitter depends on how frequently you call the **runSpeed()** function. The speed will be limited by the current value of **setMaxSpeed()**

References **_direction**, **_stepInterval**, **DIRECTION_CCW**, **DIRECTION_CW**, and **speed()**.

Referenced by **MultiStepper::moveTo()**.

◆ speed()

float AccelStepper::speed ( )

The most recently set speed.

Returns

the most recent speed in steps per second

Referenced by **setMaxSpeed()**, and **setSpeed()**.

◆ step()

void AccelStepper::step (long step) protected virtual

Called to execute a step. Only called when a new step is required. Subclasses may override to implement new stepping interfaces. The default calls **step1()**, **step2()**, **step4()** or **step8()** depending on the number of pins defined for the stepper.

Parameters

[in] **step** The current step phase number (0 to 7)

References **DRIVER**, **FULL2WIRE**, **FULL3WIRE**, **FULL4WIRE**, **FUNCTION**, **HALF3WIRE**, **HALF4WIRE**, **step0()**, **step1()**, **step2()**, **step3()**, **step4()**, **step6()**, and **step8()**.

Referenced by **runSpeed()**, **step0()**, **step1()**, **step2()**, **step3()**, **step4()**, **step6()**, **step8()**, **stepBackward()**, and **stepForward()**.

◆ step0()

void AccelStepper::step0 (long step) protected virtual

Called to execute a step using stepper functions (pins = 0) Only called when a new step is required. Calls _forward() or _backward() to perform the step

Parameters

[in] **step** The current step phase number (0 to 7)

References **step()**.

Referenced by **step()**.

◆ step1()

void AccelStepper::step1 (long step) protected virtual

Called to execute a step on a stepper driver (ie where pins == 1). Only called when a new step is required. Subclasses may override to implement new stepping interfaces. The default sets or clears the outputs of Step pin1 to step, and sets the output of _pin2 to the desired direction. The Step pin (_pin1) is pulsed for 1 microsecond which is the minimum STEP pulse width for the 3967 driver.

Parameters

[in] **step** The current step phase number (0 to 7)

References **_direction**, **setOutputPins()**, and **step()**.

Referenced by **step()**.

◆ step2()

void AccelStepper::step2 (long step) protected virtual

Called to execute a step on a 2 pin motor. Only called when a new step is required. Subclasses may override to implement new stepping interfaces. The default sets or clears the outputs of pin1 and pin2

Parameters

[in] **step** The current step phase number (0 to 7)

References **setOutputPins()**, and **step()**.

Referenced by **step()**.

◆ step3()

void AccelStepper::step3 (long step) protected virtual

Called to execute a step on a 3 pin motor, such as HDD spindle. Only called when a new step is required. Subclasses may override to implement new stepping interfaces. The default sets or clears the outputs of pin1, pin2, pin3

Parameters

`[in]` **step** The current step phase number (0 to 7)

References **setOutputPins()**, and **step()**.

Referenced by **step()**.

◆ step4()

void AccelStepper::step4 (long step) protected virtual

Called to execute a step on a 4 pin motor. Only called when a new step is required. Subclasses may override to implement new stepping interfaces. The default sets or clears the outputs of pin1, pin2, pin3, pin4.

Parameters

`[in]` **step** The current step phase number (0 to 7)

References **setOutputPins()**, and **step()**.

Referenced by **step()**.

◆ step6()

void AccelStepper::step6 (long step) protected virtual

Called to execute a step on a 3 pin motor, such as HDD spindle. Only called when a new step is required. Subclasses may override to implement new stepping interfaces. The default sets or clears the outputs of pin1, pin2, pin3

Parameters

`[in]` **step** The current step phase number (0 to 7)

References **setOutputPins()**, and **step()**.

Referenced by **step()**.

◆ step8()

void AccelStepper::step8 (long step) protected virtual

Called to execute a step on a 4 pin half-stepper motor. Only called when a new step is required. Subclasses may override to implement new stepping interfaces. The default sets or clears the outputs of pin1, pin2, pin3, pin4.

Parameters

`[in]` **step** The current step phase number (0 to 7)

References **setOutputPins()**, and **step()**.

Referenced by **step()**.

◆ stepBackward()

long AccelStepper::stepBackward ( ) protected

Called to execute a counter-clockwise(-) step. Only called when a new step is required. This decrements the _currentPos and calls **step()**

Returns

the updated current position

References **step()**.

◆ stepForward()

long AccelStepper::stepForward ( ) protected

Called to execute a clockwise(+) step. Only called when a new step is required. This increments the _currentPos and calls **step()**

Returns

the updated current position

References **step()**.

◆ stop()

void AccelStepper::stop ( )

Sets a new target position that causes the stepper to stop as quickly as possible, using the current speed and acceleration parameters.

References **move()**.

◆ targetPosition()

long AccelStepper::targetPosition ( )

The most recently set target position.

Returns

the target position in steps. Positive is clockwise from the 0 position.

Member Data Documentation

◆ _direction

boolean AccelStepper::_direction protected

Current direction motor is spinning in Protected because some peoples subclasses need it to be so

Referenced by **AccelStepper()**, **computeNewSpeed()**, **runSpeed()**, **runSpeedToPosition()**, **setSpeed()**, and **step1()**.

◆ _stepInterval

unsigned long AccelStepper::_stepInterval protected

The current interval between steps in microseconds. 0 means the motor is currently stopped with _speed == 0

Referenced by **AccelStepper()**, **computeNewSpeed()**, **runSpeed()**, **setCurrentPosition()**, and **setSpeed()**.

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

- **AccelStepper.h**
- AccelStepper.cpp

Generated by doxygen 1.9.1