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**)
void	**setAcceleration** (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)

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

void	**computeNewSpeed** ()
virtual void	**setOutputPins** (uint8_t mask)
virtual void	**step** (long step)
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**)

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 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 curent 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.

Examples:

AFMotor_ConstantSpeed.pde, AFMotor_MultiStepper.pde, Blocking.pde, Bounce.pde, ConstantSpeed.pde, MotorShield.pde, MultiStepper.pde, Overshoot.pde, ProportionalControl.pde, Quickstop.pde, and Random.pde.

Member Enumeration Documentation

enum AccelStepper::Direction protected

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

Enumerator:

DIRECTION_CCW	Clockwise.
DIRECTION_CW	Counter-Clockwise.

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::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. 1, 2, 4 or 8 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()**. **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** (pins==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** (pins==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), enableOutpuys() will be called to enable the output pins at construction time.

References **DIRECTION_CCW**, and **enableOutputs()**.

```
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_CCW.

Member Function Documentation

void AccelStepper::computeNewSpeed ( ) protected

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()

References DIRECTION_CCW, DIRECTION_CW, and distanceToGo().

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

long AccelStepper::currentPosition ( )

The currently motor position.

Returns

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

Examples:

Bounce.pde, MultiStepper.pde, Overshoot.pde, and Quickstop.pde.

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.

References **setOutputPins()**.

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.

Examples:

Bounce.pde, **MultiStepper.pde**, and **Random.pde**.

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

void AccelStepper::enableOutputs ( )

virtual

Enable motor pin outputs by setting the motor pins to OUTPUT mode. Called automatically by the constructor.

References **FULL4WIRE**, and **HALF4WIRE**.

Referenced by **AccelStepper()**.

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()**.

void AccelStepper::moveTo (long absolute)

Set the target position. The run() function will try to move the motor 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.

Examples:

Bounce.pde, MultiStepper.pde, Overshoot.pde, ProportionalControl.pde, Quickstop.pde, and Random.pde.

References computeNewSpeed().

Referenced by move(), and runToNewPosition().

boolean AccelStepper::run ( )

Poll the motor and step it if a step is due, implementing accelerations and decelerations to acheive the target position. You must call this as frequently as possible, but at least once per minimum step interval, preferably in your main loop.

Returns

true if the motor is at the target position.

Examples:

Bounce.pde, MultiStepper.pde, Overshoot.pde, Quickstop.pde, and Random.pde.

References computeNewSpeed(), and runSpeed().

Referenced by runToPosition().

boolean AccelStepper::runSpeed ( )

Poll the motor and step it if a step is due, implmenting 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.

Examples:

ConstantSpeed.pde.

References DIRECTION_CW, and step().

Referenced by run(), and runSpeedToPosition().

boolean AccelStepper::runSpeedToPosition ( )

Runs at the currently selected speed until the target position is reached Does not implement accelerations.

Returns
true if it stepped

Examples:
ProportionalControl.pde.

References DIRECTION_CCW, DIRECTION_CW, and runSpeed().

void AccelStepper::runToNewPosition (long position)

Moves the motor 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.

Examples:
Blocking.pde, and Overshoot.pde.

References moveTo(), and runToPosition().

void AccelStepper::runToPosition ( )

Moves the motor at the currently selected constant speed (forward or reverse) to the target position and blocks until it is at position. Dont use this in event loops, since it blocks.

Examples:
Quickstop.pde.

References distanceToGo(), and run().

Referenced by runToNewPosition().

void AccelStepper::setAcceleration (float acceleration)

Sets the acceleration and deceleration parameter.

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 ofthen than needed

Examples:
Blocking.pde, **Bounce.pde**, **MultiStepper.pde**, **Overshoot.pde**, **Quickstop.pde**, and **Random.pde**.

References **computeNewSpeed()**.

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.

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

void AccelStepper::setMaxSpeed (float speed)

Sets the maximum permitted speed. the run() function will accelerate up to the speed set by this function.

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.

Examples:

Blocking.pde, Bounce.pde, ConstantSpeed.pde, MultiStepper.pde, Overshoot.pde, ProportionalControl.pde, Quickstop.pde, and Random.pde.

References computeNewSpeed(), and speed().

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.

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

Examples:

MotorShield.pde.

References FULL4WIRE, and HALF4WIRE.

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

```
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

```
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

```
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.

Examples:
ConstantSpeed.pde, and **ProportionalControl.pde**.

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

float AccelStepper::speed ( )

The most recently set speed

Returns
the most recent speed in steps per second

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

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()**.

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)

Referenced by **step()**.

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 **setOutputPins()**.

Referenced by **step()**.

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()**.

Referenced by **step()**.

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()**.

Referenced by **step()**.

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()**.

Referenced by **step()**.

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()**.

Referenced by **step()**.

void AccelStepper::step8 (long step)

protected virtual

Called to execute a step on a 4 pin half-steper 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()**.

Referenced by **step()**.

void AccelStepper::stop ( )

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

Examples:

Quickstop.pde.

References **move()**.

long AccelStepper::targetPosition ( )

The most recently set target position.

Returns

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

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

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