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)
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** ()

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**)

Protected Attributes

boolean	**_direction**

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

enum AccelStepper::Direction protected

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

Enumerator	
DIRECTION_CCW	Counter-Clockwise.
DIRECTION_CW	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. 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()**. **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**, **DIRECTION_CCW**, **enableOutputs()**, and **setAcceleration()**.

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**, **DIRECTION_CCW**, and **setAcceleration()**.

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**, **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.

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

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()**.

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()**.

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()**.

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

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()**.

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 (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()**.

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 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()**.

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**, **DIRECTION_CW**, and **step()**.

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

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

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()**.

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()**.

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 ofthen than needed

References **computeNewSpeed()**.

Referenced by **AccelStepper()**.

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. Caution: the maximum speed achievable depends on your processor and clock speed.

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()**.

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

References **FULL3WIRE**, **FULL4WIRE**, **HALF3WIRE**, 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.

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

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

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 **_direction**, and **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 the current speed and acceleration parameters.

References **move()**.

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

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()**.

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

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

AccelStepper.h

```
// AccelStepper.h
//
/// \mainpage AccelStepper library for Arduino
///
/// This is the Arduino AccelStepper library.
/// It provides an object-oriented interface for 2, 3 or 4 pin stepper motors and motor drivers.
///
/// The standard Arduino IDE includes the Stepper library
/// (http://arduino.cc/en/Reference/Stepper) for stepper motors. It is
/// perfectly adequate for simple, single motor applications.
///
/// AccelStepper significantly improves on the standard Arduino Stepper library in several ways:
/// \li Supports acceleration and deceleration
/// \li Supports multiple simultaneous steppers, with independent concurrent stepping on each stepper
/// \li API functions never delay() or block
/// \li Supports 2, 3 and 4 wire steppers, plus 3 and 4 wire half steppers.
/// \li Supports alternate stepping functions to enable support of AFMotor (https://github.com/adafruit/Adafruit-Motor-Shield-libra
/// \li Supports stepper drivers such as the Sparkfun EasyDriver (based on 3967 driver chip)
/// \li Very slow speeds are supported
/// \li Extensive API
/// \li Subclass support
///
/// The latest version of this documentation can be downloaded from
/// http://www.airspayce.com/mikem/arduino/AccelStepper
/// The version of the package that this documentation refers to can be downloaded
/// from http://www.airspayce.com/mikem/arduino/AccelStepper/AccelStepper-1.57.zip
///
/// Example Arduino programs are included to show the main modes of use.
///
/// You can also find online help and discussion at http://groups.google.com/group/accelstepper
/// Please use that group for all questions and discussions on this topic.
/// Do not contact the author directly, unless it is to discuss commercial licensing.
/// Before asking a question or reporting a bug, please read
/// - http://en.wikipedia.org/wiki/Wikipedia:Reference_desk/How_to_ask_a_software_question
/// - http://www.catb.org/esr/faqs/smart-questions.html
/// - http://www.chiark.greenend.org.uk/~shgtatham/bugs.html
///
/// Tested on Arduino Diecimila and Mega with arduino-0018 & arduino-0021
/// on OpenSuSE 11.1 and avr-libc-1.6.1-1.15,
/// cross-avr-binutils-2.19-9.1, cross-avr-gcc-4.1.3_20080612-26.5.
/// Tested on Teensy http://www.pjrc.com/teensy including Teensy 3.1 built using Arduino IDE 1.0.5 with
/// teensyduino addon 1.18 and later.
///
/// \par Installation
///
/// Install in the usual way: unzip the distribution zip file to the libraries
/// sub-folder of your sketchbook.
///
/// \par Theory
///
/// This code uses speed calculations as described in
/// "Generate stepper-motor speed profiles in real time" by David Austin
/// http://fab.cba.mit.edu/classes/MIT/961.09/projects/i0/Stepper_Motor_Speed_Profile.pdf or
/// http://www.embedded.com/design/mcus-processors-and-socs/4006438/Generate-stepper-motor-speed-profiles-in-real-time or
/// http://web.archive.org/web/20140705143928/http://fab.cba.mit.edu/classes/MIT/961.09/projects/i0/Stepper_Motor_Speed_Profile.pdf
/// with the exception that AccelStepper uses steps per second rather than radians per second
/// (because we dont know the step angle of the motor)
/// An initial step interval is calculated for the first step, based on the desired acceleration
/// On subsequent steps, shorter step intervals are calculated based
/// on the previous step until max speed is achieved.
///
/// \par Adafruit Motor Shield V2
///
/// The included examples AFMotor_* are for Adafruit Motor Shield V1 and do not work with Adafruit Motor Shield V2.
/// See https://github.com/adafruit/Adafruit_Motor_Shield_V2_Library for examples that work with Adafruit Motor Shield V2.
///
/// \par Donations
///
/// This library is offered under a free GPL license for those who want to use it that way.
/// We try hard to keep it up to date, fix bugs
/// and to provide free support. If this library has helped you save time or money, please consider donating at
/// http://www.airspayce.com or here:
///
/// \htmlonly <form action="https://www.paypal.com/cgi-bin/webscr" method="post"><input type="hidden" name="cmd"
value="_donations" /> <input type="hidden" name="business" value="mikem@airspayce.com" /> <input type="hidden" name="lc"
value="AU" /> <input type="hidden" name="item_name" value="Airspayce" /> <input type="hidden" name="item_number"
value="AccelStepper" /> <input type="hidden" name="currency_code" value="USD" /> <input type="hidden" name="bn" value="PP-
DonationsBF:btn_donateCC_LG.gif:NonHosted" /> <input type="image" alt="PayPal – The safer, easier way to pay online."
name="submit" src="https://www.paypalobjects.com/en_AU/i/btn/btn_donateCC_LG.gif" /> <img alt=""
src="https://www.paypalobjects.com/en_AU/i/scr/pixel.gif" width="1" height="1" border="0" /></form> \endhtmlonly
///
/// \par Trademarks
///
/// AccelStepper is a trademark of AirSpayce Pty Ltd. The AccelStepper mark was first used on April 26 2010 for
/// international trade, and is used only in relation to motor control hardware and software.
/// It is not to be confused with any other similar marks covering other goods and services.
///
/// \par Copyright
///
/// This software is Copyright (C) 2010 Mike McCauley. Use is subject to license
/// conditions. The main licensing options available are GPL V2 or Commercial:
///
/// \par Open Source Licensing GPL V2
/// This is the appropriate option if you want to share the source code of your
/// application with everyone you distribute it to, and you also want to give them
/// the right to share who uses it. If you wish to use this software under Open
```

```
/// Source Licensing, you must contribute all your source code to the open source
/// community in accordance with the GPL Version 2 when your application is
/// distributed. See https://www.gnu.org/licenses/gpl-2.0.html
///
/// \par Commercial Licensing
/// This is the appropriate option if you are creating proprietary applications
/// and you are not prepared to distribute and share the source code of your
/// application. Purchase commercial licenses at http://airspayce.binpress.com/
///
/// \par Revision History
/// \version 1.0 Initial release
///
/// \version 1.1 Added speed() function to get the current speed.
/// \version 1.2 Added runSpeedToPosition() submitted by Gunnar Arndt.
/// \version 1.3 Added support for stepper drivers (ie with Step and Direction inputs) with _pins == 1
/// \version 1.4 Added functional contructor to support AFMotor, contributed by Limor, with example sketches.
/// \version 1.5 Improvements contributed by Peter Mousley: Use of microsecond steps and other speed improvements
///              to increase max stepping speed to about 4kHz. New option for user to set the min allowed pulse width.
///              Added checks for already running at max speed and skip further calcs if so.
/// \version 1.6 Fixed a problem with wrapping of microsecond stepping that could cause stepping to hang.
///              Reported by Sandy Noble.
///              Removed redundant _lastRunTime member.
/// \version 1.7 Fixed a bug where setCurrentPosition() did not always work as expected.
///              Reported by Peter Linhart.
/// \version 1.8 Added support for 4 pin half-steppers, requested by Harvey Moon
/// \version 1.9 setCurrentPosition() now also sets motor speed to 0.
/// \version 1.10 Builds on Arduino 1.0
/// \version 1.11 Improvments from Michael Ellison:
///   Added optional enable line support for stepper drivers
///   Added inversion for step/direction/enable lines for stepper drivers
/// \version 1.12 Announce Google Group
/// \version 1.13 Improvements to speed calculation. Cost of calculation is now less in the worst case,
///    and more or less constant in all cases. This should result in slightly beter high speed performance, and
///    reduce anomalous speed glitches when other steppers are accelerating.
///    However, its hard to see how to replace the sqrt() required at the very first step from 0 speed.
/// \version 1.14 Fixed a problem with compiling under arduino 0021 reported by EmbeddedMan
/// \version 1.15 Fixed a problem with runSpeedToPosition which did not correctly handle
///    running backwards to a smaller target position. Added examples
/// \version 1.16 Fixed some cases in the code where abs() was used instead of fabs().
/// \version 1.17 Added example ProportionalControl
/// \version 1.18 Fixed a problem: If one calls the funcion runSpeed() when Speed is zero, it makes steps
///    without counting. reported by  Friedrich, Klappenbach.
/// \version 1.19 Added MotorInterfaceType and symbolic names for the number of pins to use
///               for the motor interface. Updated examples to suit.
///               Replaced individual pin assignment variables _pin1, _pin2 etc with array _pin[4].
///               _pins member changed to _interface.
///               Added _pinInverted array to simplify pin inversion operations.
///               Added new function setOutputPins() which sets the motor output pins.
///               It can be overridden in order to provide, say, serial output instead of parallel output
///               Some refactoring and code size reduction.
/// \version 1.20 Improved documentation and examples to show need for correctly
///               specifying AccelStepper::FULL4WIRE and friends.
/// \version 1.21 Fixed a problem where desiredSpeed could compute the wrong step acceleration
///               when _speed was small but non-zero. Reported by Brian Schmalz.
///               Precompute sqrt_twoa to improve performance and max possible stepping speed
/// \version 1.22 Added Bounce.pde example
///               Fixed a problem where calling moveTo(), setMaxSpeed(), setAcceleration() more
///               frequently than the step time, even
///               with the same values, would interfere with speed calcs. Now a new speed is computed
///               only if there was a change in the set value. Reported by Brian Schmalz.
/// \version 1.23 Rewrite of the speed algorithms in line with
///               http://fab.cba.mit.edu/classes/MIT/961.09/projects/i0/Stepper_Motor_Speed_Profile.pdf
///               Now expect smoother and more linear accelerations and decelerations. The desiredSpeed()
///               function was removed.
/// \version 1.24  Fixed a problem introduced in 1.23: with runToPosition, which did never returned
/// \version 1.25  Now ignore attempts to set acceleration to 0.0
/// \version 1.26  Fixed a problem where certina combinations of speed and accelration could cause
///                oscillation about the target position.
/// \version 1.27  Added stop() function to stop as fast as possible with current acceleration parameters.
///                Also added new Quickstop example showing its use.
/// \version 1.28  Fixed another problem where certain combinations of speed and accelration could cause
///                oscillation about the target position.
///                Added support for 3 wire full and half steppers such as Hard Disk Drive spindle.
///                Contributed by Yuri Ivatchkovitch.
/// \version 1.29  Fixed a problem that could cause a DRIVER stepper to continually step
///                with some sketches. Reported by Vadim.
/// \version 1.30  Fixed a problem that could cause stepper to back up a few steps at the end of
///                accelerated travel with certain speeds. Reported and patched by jolo.
/// \version 1.31  Updated author and distribution location details to airspayce.com
/// \version 1.32  Fixed a problem with enableOutputs() and setEnablePin on Arduino Due that
///                prevented the enable pin changing stae correctly. Reported by Duane Bishop.
/// \version 1.33  Fixed an error in example AFMotor_ConstantSpeed.pde did not setMaxSpeed();
///                Fixed a problem that caused incorrect pin sequencing of FULL3WIRE and HALF3WIRE.
///                Unfortunately this meant changing the signature for all step*() functions.
///                Added example MotorShield, showing how to use AdaFruit Motor Shield to control
///                a 3 phase motor such as a HDD spindle motor (and without using the AFMotor library.
/// \version 1.34  Added setPinsInverted(bool pin1Invert, bool pin2Invert, bool pin3Invert, bool pin4Invert, bool enableInvert)
///                to allow inversion of 2, 3 and 4 wire stepper pins. Requested by Oleg.
/// \version 1.35  Removed default args from setPinsInverted(bool, bool, bool, bool, bool) to prevent ambiguity with
///                setPinsInverted(bool, bool, bool). Reported by Mac Mac.
/// \version 1.36  Changed enableOutputs() and disableOutputs() to be virtual so can be overridden.
///                Added new optional argument 'enable' to constructor, which allows you toi disable the
///                automatic enabling of outputs at construction time. Suggested by Guido.
/// \version 1.37  Fixed a problem with step1 that could cause a rogue step in the
///                wrong direction (or not,
///                depending on the setup-time requirements of the connected hardware).
///                Reported by Mark Tillotson.
/// \version 1.38  run() function incorrectly always returned true. Updated function and doc so it returns true
///                if the motor is still running to the target position.
/// \version 1.39  Updated typos in keywords.txt, courtesey Jon Magill.
/// \version 1.40  Updated documentation, including testing on Teensy 3.1
/// \version 1.41  Fixed an error in the acceleration calculations, resulting in acceleration of haldf the intended value
```

```
193 /// \version 1.42  Improved support for FULL3WIRE and HALF3WIRE output pins. These changes were in Yuri's original
194 ///                contribution but did not make it into production.<br>
195 /// \version 1.43  Added DualMotorShield example. Shows how to use AccelStepper to control 2 x 2 phase steppers using the
196 ///                Itead Studio Arduino Dual Stepper Motor Driver Shield model IM120417015.<br>
197 /// \version 1.44  examples/DualMotorShield/DualMotorShield.ino examples/DualMotorShield/DualMotorShield.pde
198 ///                was missing from the distribution.<br>
199 /// \version 1.45  Fixed a problem where if setAcceleration was not called, there was no default
200 ///                acceleration. Reported by Michael Newman.<br>
201 /// \version 1.45  Fixed inaccuracy in acceleration rate by using Equation 15, suggested by Sebastian Gracki.<br>
202 ///                Performance improvements in runSpeed suggested by Jaakko Fagerlund.<br>
203 /// \version 1.46  Fixed error in documentation for runToPosition().
204 ///                Reinstated time calculations in runSpeed() since new version is reported
205 ///                not to work correctly under some circumstances. Reported by Oleg V Gavva.<br>
206 /// \version 1.48  2015-08-25
207 ///                Added new class MultiStepper that can manage multiple AccelSteppers,
208 ///                and cause them all to move
209 ///                to selected positions at such a (constant) speed that they all arrive at their
210 ///                target position at the same time. Suitable for X-Y flatbeds etc.<br>
211 ///                Added new method maxSpeed() to AccelStepper to return the currently configured maxSpeed.<br>
212 /// \version 1.49  2016-01-02
213 ///                Testing with VID28 series instrument stepper motors and EasyDriver.
214 ///                OK, although with light pointers
215 ///                and slow speeds like 180 full steps per second the motor movement can be erratic,
216 ///                probably due to some mechanical resonance. Best to accelerate through this speed.<br>
217 ///                Added isRunning().<br>
218 /// \version 1.50 2016-02-25
219 ///                AccelStepper::disableOutputs now sets the enable pion to OUTPUT mode if the enable pin is defined.
220 ///                Patch from Piet De Jong.<br>
221 ///                Added notes about the fact that AFMotor_* examples do not work with Adafruit Motor Shield V2.<br>
222 /// \version 1.51 2016-03-24
223 ///                Fixed a problem reported by gregor: when resetting the stepper motor position using setCurrentPosition() the
224 ///                stepper speed is reset by setting _stepInterval to 0, but _speed is not
225 ///                reset. this results in the stepper motor not starting again when calling
226 ///                setSpeed() with the same speed the stepper was set to before.
227 /// \version 1.52 2016-08-09
228 ///                Added MultiStepper to keywords.txt.
229 ///                Improvements to efficiency of AccelStepper::runSpeed() as suggested by David Grayson.
230 ///                Improvements to speed accuracy as suggested by David Grayson.
231 /// \version 1.53 2016-08-14
232 ///                Backed out Improvements to speed accuracy from 1.52 as it did not work correctly.
233 /// \version 1.54 2017-01-24
234 ///                Fixed some warnings about unused arguments.
235 /// \version 1.55 2017-01-25
236 ///                Fixed another warning in MultiStepper.cpp
237 /// \version 1.56 2017-02-03
238 ///                Fixed minor documentation error with DIRECTION_CCW and DIRECTION_CW. Reported by David Mutterer.
239 ///                Added link to Binpress commercial license purchasing.
240 /// \version 1.57 2017-03-28
241 ///                _direction moved to protected at the request of Rudy Ercek.
242 ///                setMaxSpeed() and setAcceleration() now correct negative values to be positive.
243 ///
244 /// \author  Mike McCauley (mikem@airspayce.com) DO NOT CONTACT THE AUTHOR DIRECTLY: USE THE LISTS
245 // Copyright (C) 2009-2013 Mike McCauley
246 // $Id: AccelStepper.h,v 1.27 2016/08/14 10:26:54 mikem Exp mikem $
247
248 #ifndef AccelStepper_h
249 #define AccelStepper_h
250
251 #include <stdlib.h>
252 #if ARDUINO >= 100
253 #include <Arduino.h>
254 #else
255 #include <WProgram.h>
256 #include <wiring.h>
257 #endif
258
259 // These defs cause trouble on some versions of Arduino
260 #undef round
261
262 ////////////////////////////////////////////////////////////////////
263 /// \class AccelStepper AccelStepper.h <AccelStepper.h>
264 /// \brief Support for stepper motors with acceleration etc.
265 ///
266 /// This defines a single 2 or 4 pin stepper motor, or stepper moter with fdriver chip, with optional
267 /// acceleration, deceleration, absolute positioning commands etc. Multiple
268 /// simultaneous steppers are supported, all moving
269 /// at different speeds and accelerations.
270 ///
271 /// \par Operation
272 /// This module operates by computing a step time in microseconds. The step
273 /// time is recomputed after each step and after speed and acceleration
274 /// parameters are changed by the caller. The time of each step is recorded in
275 /// microseconds. The run() function steps the motor once if a new step is due.
276 /// The run() function must be called frequently until the motor is in the
277 /// desired position, after which time run() will do nothing.
278 ///
279 /// \par Positioning
280 /// Positions are specified by a signed long integer. At
281 /// construction time, the current position of the motor is consider to be 0. Positive
282 /// positions are clockwise from the initial position; negative positions are
283 /// anticlockwise. The current position can be altered for instance after
284 /// initialization positioning.
285 ///
286 /// \par Caveats
287 /// This is an open loop controller: If the motor stalls or is oversped,
288 /// AccelStepper will not have a correct
289 /// idea of where the motor really is (since there is no feedback of the motor's
290 /// real position. We only know where we _think_ it is, relative to the
291 /// initial starting point).
292 ///
293 /// \par Performance
294 /// The fastest motor speed that can be reliably supported is about 4000 steps per
```

```
295 /// second at a clock frequency of 16 MHz on Arduino such as Uno etc.
296 /// Faster processors can support faster stepping speeds.
297 /// However, any speed less than that
298 /// down to very slow speeds (much less than one per second) are also supported,
299 /// provided the run() function is called frequently enough to step the motor
300 /// whenever required for the speed set.
301 /// Calling setAcceleration() is expensive,
302 /// since it requires a square root to be calculated.
303 ///
304 /// Gregor Christandl reports that with an Arduino Due and a simple test program,
305 /// he measured 43163 steps per second using runSpeed(),
306 /// and 16214 steps per second using run();
307 class AccelStepper
308 {
309 public:
310     /// \brief Symbolic names for number of pins.
311     /// Use this in the pins argument the AccelStepper constructor to
312     /// provide a symbolic name for the number of pins
313     /// to use.
314     typedef enum
315     {
316         FUNCTION  = 0, ///< Use the functional interface, implementing your own driver functions (internal use only)
317         DRIVER    = 1, ///< Stepper Driver, 2 driver pins required
318         FULL2WIRE = 2, ///< 2 wire stepper, 2 motor pins required
319         FULL3WIRE = 3, ///< 3 wire stepper, such as HDD spindle, 3 motor pins required
320         FULL4WIRE = 4, ///< 4 wire full stepper, 4 motor pins required
321         HALF3WIRE = 6, ///< 3 wire half stepper, such as HDD spindle, 3 motor pins required
322         HALF4WIRE = 8  ///< 4 wire half stepper, 4 motor pins required
323     } MotorInterfaceType;
324
325     /// Constructor. You can have multiple simultaneous steppers, all moving
326     /// at different speeds and accelerations, provided you call their run()
327     /// functions at frequent enough intervals. Current Position is set to 0, target
328     /// position is set to 0. MaxSpeed and Acceleration default to 1.0.
329     /// The motor pins will be initialised to OUTPUT mode during the
330     /// constructor by a call to enableOutputs().
331     /// \param[in] interface Number of pins to interface to. Integer values are
332     /// supported, but it is preferred to use the \ref MotorInterfaceType symbolic names.
333     /// AccelStepper::DRIVER (1) means a stepper driver (with Step and Direction pins).
334     /// If an enable line is also needed, call setEnablePin() after construction.
335     /// You may also invert the pins using setPinsInverted().
336     /// AccelStepper::FULL2WIRE (2) means a 2 wire stepper (2 pins required).
337     /// AccelStepper::FULL3WIRE (3) means a 3 wire stepper, such as HDD spindle (3 pins required).
338     /// AccelStepper::FULL4WIRE (4) means a 4 wire stepper (4 pins required).
339     /// AccelStepper::HALF3WIRE (6) means a 3 wire half stepper, such as HDD spindle (3 pins required)
340     /// AccelStepper::HALF4WIRE (8) means a 4 wire half stepper (4 pins required)
341     /// Defaults to AccelStepper::FULL4WIRE (4) pins.
342     /// \param[in] pin1 Arduino digital pin number for motor pin 1. Defaults
343     /// to pin 2. For a AccelStepper::DRIVER (interface==1),
344     /// this is the Step input to the driver. Low to high transition means to step)
345     /// \param[in] pin2 Arduino digital pin number for motor pin 2. Defaults
346     /// to pin 3. For a AccelStepper::DRIVER (interface==1),
347     /// this is the Direction input the driver. High means forward.
348     /// \param[in] pin3 Arduino digital pin number for motor pin 3. Defaults
349     /// to pin 4.
350     /// \param[in] pin4 Arduino digital pin number for motor pin 4. Defaults
351     /// to pin 5.
352     /// \param[in] enable If this is true (the default), enableOutputs() will be called to enable
353     /// the output pins at construction time.
354     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);
355
356     /// Alternate Constructor which will call your own functions for forward and backward steps.
357     /// You can have multiple simultaneous steppers, all moving
358     /// at different speeds and accelerations, provided you call their run()
359     /// functions at frequent enough intervals. Current Position is set to 0, target
360     /// position is set to 0. MaxSpeed and Acceleration default to 1.0.
361     /// Any motor initialization should happen before hand, no pins are used or initialized.
362     /// \param[in] forward void-returning procedure that will make a forward step
363     /// \param[in] backward void-returning procedure that will make a backward step
364     AccelStepper(void (*forward)(), void (*backward)());
365
366     /// Set the target position. The run() function will try to move the motor (at most one step per call)
367     /// from the current position to the target position set by the most
368     /// recent call to this function. Caution: moveTo() also recalculates the speed for the next step.
369     /// If you are trying to use constant speed movements, you should call setSpeed() after calling moveTo().
370     /// \param[in] absolute The desired absolute position. Negative is
371     /// anticlockwise from the 0 position.
372     void    moveTo(long absolute);
373
374     /// Set the target position relative to the current position
375     /// \param[in] relative The desired position relative to the current position. Negative is
376     /// anticlockwise from the current position.
377     void    move(long relative);
378
379     /// Poll the motor and step it if a step is due, implementing
380     /// accelerations and decelerations to acheive the target position. You must call this as
381     /// frequently as possible, but at least once per minimum step time interval,
382     /// 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,
383     /// based on the current speed and the time since the last step.
384     /// \return true if the motor is still running to the target position.
385     boolean run();
386
387     /// Poll the motor and step it if a step is due, implementing a constant
388     /// speed as set by the most recent call to setSpeed(). You must call this as
389     /// frequently as possible, but at least once per step interval,
390     /// \return true if the motor was stepped.
391     boolean runSpeed();
392
393     /// Sets the maximum permitted speed. The run() function will accelerate
394     /// up to the speed set by this function.
395     /// Caution: the maximum speed achievable depends on your processor and clock speed.
```

```
396     /// \param[in] speed The desired maximum speed in steps per second. Must
397     /// be > 0. Caution: Speeds that exceed the maximum speed supported by the processor may
398     /// Result in non-linear accelerations and decelerations.
399     void    setMaxSpeed(float speed);
400
401     /// returns the maximum speed configured for this stepper
402     /// that was previously set by setMaxSpeed();
403     /// \return The currently configured maximum speed
404     float   maxSpeed();
405
406     /// Sets the acceleration/deceleration rate.
407     /// \param[in] acceleration The desired acceleration in steps per second
408     /// per second. Must be > 0.0. This is an expensive call since it requires a square
409     /// root to be calculated. Dont call more ofthen than needed
410     void    setAcceleration(float acceleration);
411
412     /// Sets the desired constant speed for use with runSpeed().
413     /// \param[in] speed The desired constant speed in steps per
414     /// second. Positive is clockwise. Speeds of more than 1000 steps per
415     /// second are unreliable. Very slow speeds may be set (eg 0.00027777 for
416     /// once per hour, approximately. Speed accuracy depends on the Arduino
417     /// crystal. Jitter depends on how frequently you call the runSpeed() function.
418     void    setSpeed(float speed);
419
420     /// The most recently set speed
421     /// \return the most recent speed in steps per second
422     float   speed();
423
424     /// The distance from the current position to the target position.
425     /// \return the distance from the current position to the target position
426     /// in steps. Positive is clockwise from the current position.
427     long    distanceToGo();
428
429     /// The most recently set target position.
430     /// \return the target position
431     /// in steps. Positive is clockwise from the 0 position.
432     long    targetPosition();
433
434     /// The currently motor position.
435     /// \return the current motor position
436     /// in steps. Positive is clockwise from the 0 position.
437     long    currentPosition();
438
439     /// Resets the current position of the motor, so that wherever the motor
440     /// happens to be right now is considered to be the new 0 position. Useful
441     /// for setting a zero position on a stepper after an initial hardware
442     /// positioning move.
443     /// Has the side effect of setting the current motor speed to 0.
444     /// \param[in] position The position in steps of wherever the motor
445     /// happens to be right now.
446     void    setCurrentPosition(long position);
447
448     /// Moves the motor (with acceleration/deceleration)
449     /// to the target position and blocks until it is at
450     /// position. Dont use this in event loops, since it blocks.
451     void    runToPosition();
452
453     /// Runs at the currently selected speed until the target position is reached
454     /// Does not implement accelerations.
455     /// \return true if it stepped
456     boolean runSpeedToPosition();
457
458     /// Moves the motor (with acceleration/deceleration)
459     /// to the new target position and blocks until it is at
460     /// position. Dont use this in event loops, since it blocks.
461     /// \param[in] position The new target position.
462     void    runToNewPosition(long position);
463
464     /// Sets a new target position that causes the stepper
465     /// to stop as quickly as possible, using the current speed and acceleration parameters.
466     void stop();
467
468     /// Disable motor pin outputs by setting them all LOW
469     /// Depending on the design of your electronics this may turn off
470     /// the power to the motor coils, saving power.
471     /// This is useful to support Arduino low power modes: disable the outputs
472     /// during sleep and then reenable with enableOutputs() before stepping
473     /// again.
474     /// If the enable Pin is defined, sets it to OUTPUT mode and clears the pin to disabled.
475     virtual void    disableOutputs();
476
477     /// Enable motor pin outputs by setting the motor pins to OUTPUT
478     /// mode. Called automatically by the constructor.
479     /// If the enable Pin is defined, sets it to OUTPUT mode and sets the pin to enabled.
480     virtual void    enableOutputs();
481
482     /// Sets the minimum pulse width allowed by the stepper driver. The minimum practical pulse width is
483     /// approximately 20 microseconds. Times less than 20 microseconds
484     /// will usually result in 20 microseconds or so.
485     /// \param[in] minWidth The minimum pulse width in microseconds.
486     void    setMinPulseWidth(unsigned int minWidth);
487
488     /// Sets the enable pin number for stepper drivers.
489     /// 0xFF indicates unused (default).
490     /// Otherwise, if a pin is set, the pin will be turned on when
491     /// enableOutputs() is called and switched off when disableOutputs()
492     /// is called.
493     /// \param[in] enablePin Arduino digital pin number for motor enable
494     /// \sa setPinsInverted
495     void    setEnablePin(uint8_t enablePin = 0xff);
496
497     /// Sets the inversion for stepper driver pins
```

```
498     /// \param[in] directionInvert True for inverted direction pin, false for non-inverted
499     /// \param[in] stepInvert      True for inverted step pin, false for non-inverted
500     /// \param[in] enableInvert    True for inverted enable pin, false (default) for non-inverted
501     void    setPinsInverted(bool directionInvert = false, bool stepInvert = false, bool enableInvert = false);
502
503     /// Sets the inversion for 2, 3 and 4 wire stepper pins
504     /// \param[in] pin1Invert True for inverted pin1, false for non-inverted
505     /// \param[in] pin2Invert True for inverted pin2, false for non-inverted
506     /// \param[in] pin3Invert True for inverted pin3, false for non-inverted
507     /// \param[in] pin4Invert True for inverted pin4, false for non-inverted
508     /// \param[in] enableInvert    True for inverted enable pin, false (default) for non-inverted
509     void    setPinsInverted(bool pin1Invert, bool pin2Invert, bool pin3Invert, bool pin4Invert, bool enableInvert);
510
511     /// Checks to see if the motor is currently running to a target
512     /// \return true if the speed is not zero or not at the target position
513     bool    isRunning();
514
515 protected:
516
517     /// \brief Direction indicator
518     /// Symbolic names for the direction the motor is turning
519     typedef enum
520     {
521         DIRECTION_CCW = 0,  ///< Counter-Clockwise
522         DIRECTION_CW  = 1   ///< Clockwise
523     } Direction;
524
525     /// Forces the library to compute a new instantaneous speed and set that as
526     /// the current speed. It is called by
527     /// the library:
528     /// \li  after each step
529     /// \li  after change to maxSpeed through setMaxSpeed()
530     /// \li  after change to acceleration through setAcceleration()
531     /// \li  after change to target position (relative or absolute) through
532     /// move() or moveTo()
533     void            computeNewSpeed();
534
535     /// Low level function to set the motor output pins
536     /// bit 0 of the mask corresponds to _pin[0]
537     /// bit 1 of the mask corresponds to _pin[1]
538     /// You can override this to impment, for example serial chip output insted of using the
539     /// output pins directly
540     virtual void   setOutputPins(uint8_t mask);
541
542     /// Called to execute a step. Only called when a new step is
543     /// required. Subclasses may override to implement new stepping
544     /// interfaces. The default calls step1(), step2(), step4() or step8() depending on the
545     /// number of pins defined for the stepper.
546     /// \param[in] step The current step phase number (0 to 7)
547     virtual void   step(long step);
548
549     /// Called to execute a step using stepper functions (pins = 0) Only called when a new step is
550     /// required. Calls _forward() or _backward() to perform the step
551     /// \param[in] step The current step phase number (0 to 7)
552     virtual void   step0(long step);
553
554     /// Called to execute a step on a stepper driver (ie where pins == 1). Only called when a new step is
555     /// required. Subclasses may override to implement new stepping
556     /// interfaces. The default sets or clears the outputs of Step pin1 to step,
557     /// and sets the output of _pin2 to the desired direction. The Step pin (_pin1) is pulsed for 1 microsecond
558     /// which is the minimum STEP pulse width for the 3967 driver.
559     /// \param[in] step The current step phase number (0 to 7)
560     virtual void   step1(long step);
561
562     /// Called to execute a step on a 2 pin motor. Only called when a new step is
563     /// required. Subclasses may override to implement new stepping
564     /// interfaces. The default sets or clears the outputs of pin1 and pin2
565     /// \param[in] step The current step phase number (0 to 7)
566     virtual void   step2(long step);
567
568     /// Called to execute a step on a 3 pin motor, such as HDD spindle. Only called when a new step is
569     /// required. Subclasses may override to implement new stepping
570     /// interfaces. The default sets or clears the outputs of pin1, pin2,
571     /// pin3
572     /// \param[in] step The current step phase number (0 to 7)
573     virtual void   step3(long step);
574
575     /// Called to execute a step on a 4 pin motor. Only called when a new step is
576     /// required. Subclasses may override to implement new stepping
577     /// interfaces. The default sets or clears the outputs of pin1, pin2,
578     /// pin3, pin4.
579     /// \param[in] step The current step phase number (0 to 7)
580     virtual void   step4(long step);
581
582     /// Called to execute a step on a 3 pin motor, such as HDD spindle. Only called when a new step is
583     /// required. Subclasses may override to implement new stepping
584     /// interfaces. The default sets or clears the outputs of pin1, pin2,
585     /// pin3
586     /// \param[in] step The current step phase number (0 to 7)
587     virtual void   step6(long step);
588
589     /// Called to execute a step on a 4 pin half-steper motor. Only called when a new step is
590     /// required. Subclasses may override to implement new stepping
591     /// interfaces. The default sets or clears the outputs of pin1, pin2,
592     /// pin3, pin4.
593     /// \param[in] step The current step phase number (0 to 7)
594     virtual void   step8(long step);
595
596     /// Current direction motor is spinning in
597     /// Protected because some peoples subclasses need it to be so
598     boolean _direction; // 1 == CW
599
```

```
600 private:
601     /// Number of pins on the stepper motor. Permits 2 or 4. 2 pins is a
602     /// bipolar, and 4 pins is a unipolar.
603     uint8_t        _interface;          // 0, 1, 2, 4, 8, See MotorInterfaceType
604 
605     /// Arduino pin number assignments for the 2 or 4 pins required to interface to the
606     /// stepper motor or driver
607     uint8_t        _pin[4];
608 
609     /// Whether the _pins is inverted or not
610     uint8_t        _pinInverted[4];
611 
612     /// The current absolution position in steps.
613     long           _currentPos;    // Steps
614 
615     /// The target position in steps. The AccelStepper library will move the
616     /// motor from the _currentPos to the _targetPos, taking into account the
617     /// max speed, acceleration and deceleration
618     long           _targetPos;     // Steps
619 
620     /// The current motos speed in steps per second
621     /// Positive is clockwise
622     float          _speed;         // Steps per second
623 
624     /// The maximum permitted speed in steps per second. Must be > 0.
625     float          _maxSpeed;
626 
627     /// The acceleration to use to accelerate or decelerate the motor in steps
628     /// per second per second. Must be > 0
629     float          _acceleration;
630     float          _sqrt_twoa; // Precomputed sqrt(2*_acceleration)
631 
632     /// The current interval between steps in microseconds.
633     /// 0 means the motor is currently stopped with _speed == 0
634     unsigned long  _stepInterval;
635 
636     /// The last step time in microseconds
637     unsigned long  _lastStepTime;
638 
639     /// The minimum allowed pulse width in microseconds
640     unsigned int   _minPulseWidth;
641 
642     /// Is the direction pin inverted?
643     ///bool           _dirInverted; /// Moved to _pinInverted[1]
644 
645     /// Is the step pin inverted?
646     ///bool           _stepInverted; /// Moved to _pinInverted[0]
647 
648     /// Is the enable pin inverted?
649     bool           _enableInverted;
650 
651     /// Enable pin for stepper driver, or 0xFF if unused.
652     uint8_t        _enablePin;
653 
654     /// The pointer to a forward-step procedure
655     void (*_forward)();
656 
657     /// The pointer to a backward-step procedure
658     void (*_backward)();
659 
660     /// The step counter for speed calculations
661     long _n;
662 
663     /// Initial step size in microseconds
664     float _c0;
665 
666     /// Last step size in microseconds
667     float _cn;
668 
669     /// Min step size in microseconds based on maxSpeed
670     float _cmin; // at max speed
671 
672 };
673 
674 /// @example Random.pde
675 /// Make a single stepper perform random changes in speed, position and acceleration
676 
677 /// @example Overshoot.pde
678 ///  Check overshoot handling
679 /// which sets a new target position and then waits until the stepper has
680 /// achieved it. This is used for testing the handling of overshoots
681 
682 /// @example MultipleSteppers.pde
683 /// Shows how to multiple simultaneous steppers
684 /// Runs one stepper forwards and backwards, accelerating and decelerating
685 /// at the limits. Runs other steppers at the same time
686 
687 /// @example ConstantSpeed.pde
688 /// Shows how to run AccelStepper in the simplest,
689 /// fixed speed mode with no accelerations
690 
691 /// @example Blocking.pde
692 /// Shows how to use the blocking call runToNewPosition
693 /// Which sets a new target position and then waits until the stepper has
694 /// achieved it.
695 
696 /// @example AFMotor_MultiStepper.pde
697 /// Control both Stepper motors at the same time with different speeds
698 /// and accelerations.
699 
700 /// @example AFMotor_ConstantSpeed.pde
701 /// Shows how to run AccelStepper in the simplest,
```

```
702 /// fixed speed mode with no accelerations
703
704 /// @example ProportionalControl.pde
705 /// Make a single stepper follow the analog value read from a pot or whatever
706 /// The stepper will move at a constant speed to each newly set posiiton,
707 /// depending on the value of the pot.
708
709 /// @example Bounce.pde
710 /// Make a single stepper bounce from one limit to another, observing
711 /// accelrations at each end of travel
712
713 /// @example Quickstop.pde
714 /// Check stop handling.
715 /// Calls stop() while the stepper is travelling at full speed, causing
716 /// the stepper to stop as quickly as possible, within the constraints of the
717 /// current acceleration.
718
719 /// @example MotorShield.pde
720 /// Shows how to use AccelStepper to control a 3-phase motor, such as a HDD spindle motor
721 /// using the Adafruit Motor Shield http://www.ladyada.net/make/mshield/index.html.
722
723 /// @example DualMotorShield.pde
724 /// Shows how to use AccelStepper to control 2 x 2 phase steppers using the
725 /// Itead Studio Arduino Dual Stepper Motor Driver Shield
726 /// model IM120417015
727
728 #endif
```

MultiStepper.h

```
1  // MultiStepper.h
2
3  #ifndef MultiStepper_h
4  #define MultiStepper_h
5
6  #include <stdlib.h>
7  #if ARDUINO >= 100
8  #include <Arduino.h>
9  #else
10 #include <WProgram.h>
11 #include <wiring.h>
12 #endif
13
14 #define MULTISTEPPER_MAX_STEPPERS 10
15
16 class AccelStepper;
17
18 ////////////////////////////////////////////////////////////////////
19 /// \class MultiStepper MultiStepper.h <MultiStepper.h>
20 /// \brief Operate multiple AccelSteppers in a co-ordinated fashion
21 ///
22 /// This class can manage multiple AccelSteppers (up to MULTISTEPPER_MAX_STEPPERS = 10),
23 /// and cause them all to move
24 /// to selected positions at such a (constant) speed that they all arrive at their
25 /// target position at the same time. This can be used to support devices with multiple steppers
26 /// on say multiple axes to cause linear diagonal motion. Suitable for use with X-Y plotters, flatbeds,
27 /// 3D printers etc
28 /// to get linear straight line movement between arbitrary 2d (or 3d or ...) positions.
29 ///
30 /// Caution: only constant speed stepper motion is supported: acceleration and deceleration is not supported
31 /// All the steppers managed by MultiStepper will step at a constant speed to their
32 /// target (albeit perhaps different speeds for each stepper).
33 class MultiStepper
34 {
35 public:
36     /// Constructor
37     MultiStepper();
38
39     /// Add a stepper to the set of managed steppers
40     /// There is an upper limit of MULTISTEPPER_MAX_STEPPERS = 10 to the number of steppers that can be managed
41     /// \param[in] stepper Reference to a stepper to add to the managed list
42     /// \return true if successful. false if the number of managed steppers would exceed MULTISTEPPER_MAX_STEPPERS
43     boolean addStepper(AccelStepper& stepper);
44
45     /// Set the target positions of all managed steppers
46     /// according to a coordinate array.
47     /// New speeds will be computed for each stepper so they will all arrive at their
48     /// respective targets at very close to the same time.
49     /// \param[in] absolute An array of desired absolute stepper positions. absolute[0] will be used to set
50     /// the absolute position of the first stepper added by addStepper() etc. The array must be at least as
   long as
51     /// the number of steppers that have been added by addStepper, else results are undefined.
52     void moveTo(long absolute[]);
53
54     /// Calls runSpeed() on all the managed steppers
55     /// that have not acheived their target position.
56     /// \return true if any stepper is still in the process of running to its target position.
57     boolean run();
58
59     /// Runs all managed steppers until they acheived their target position.
60     /// Blocks until all that position is acheived. If you dont
61     /// want blocking consider using run() instead.
62     void    runSpeedToPosition();
63
64 private:
65     /// Array of pointers to the steppers we are controlling.
66     /// Fills from 0 onwards
67     AccelStepper* _steppers[MULTISTEPPER_MAX_STEPPERS];
68
69     /// Number of steppers we are controlling and the number
70     /// of steppers in _steppers[]
71     uint8_t       _num_steppers;
72 };
73
74 /// @example MultiStepper.pde
75 /// Use MultiStepper class to manage multiple steppers and make them all move to
76 /// the same position at the same time for linear 2d (or 3d) motion.
77
78 #endif
```