MobaTools – a Arduino Library for Model Railroaders

Documentation for Version 0.6

The MobaTools are a compilation of classes that are useful in the model railroad environment. They facilitate some standard tasks, thus enabling even beginners to write complex Sketches.

Summary of the classes:

Servo8 Control of up to 16 servos. The methods are largely compatible with the standard Servolib of the Arduino. However, the speed of movement of the servo can be specified..

Stepper4 Control of up to 4 unipolar stepper motors. Unlike the standard library 'steppers' the calls are non blocking. Once a reference point has been established, the stepper motor can be positioned absolutely (in degrees) with nearly the same methods as in the Servo8 Library.

SoftLed Soft switching of LEDs. As connecting pins all pins are permitted, where the 'analog write' call of the Arduino works.

EggTimer With this class it is easy to create time delays without blocking the Sketch.

The MobaTools intensively use the timer1 of Arduino. Therefor all other libraries or functions that also use the timer1 cannot be used simultaneously. This concerns, for example, the standard ServoLib and analog write on pins 9 + 10.

Klasse Servo8

Creating one Servo object (up to 16 objects are allowed):

```
Servo8     myServo;
```

Functions:

```
byte myServo.attach( int pin );
```

Assigns a pin where the pulses are output. The pin is switched to output, but no pulses are produced yet.

```
byte myServo.attach( int pin, bool autoOff );
```

If the optional parameter 'AutoOff' with the value 'true' is passed, then the pulse is turned off automatically when the length of the pulse does not change for more than 1sec.

```
byte myServo.attach( int pin, int pos0, int pos180, bool autoOff );
```

With the parameters pos0 and pos180 can be specified, which pulse lengths are assigned to the angles '0' or '180'. (The parameter AutoOff is optional)

```
byte myServo.detach();
```

The assignment of the servos to the pin is removed, it will no longer generates pulses.

```
void myServo.setSpeed(int Speed);
```

Set moving speed of the servos. 'Speed', is the value (in 0,5µs units) the pulse length changes every 20ms until target length. Speed = 1 means that 40s are needed to change the

pulse width from 1ms to 2ms.
Speed = 0 (default value) means direct change of pulselength (like standard servo Library)

```
void myServo.write(int angle);
```

Specify arget position. The servo moves with the preset speed to that position. With values from 0 ... 180 the value is interpreted as an angle, and accordingly converted into a pulse length. Values > 180 are interpreted as pulse length in microseconds, where pulse length is limited to the minimum (700µs) or maximum (2300µs).
The write command takes effect immediately, even if the servo has not yet reached its set position of the previous write.
If this is the first 'write' call after an 'attach', the specified pulse length is output immediately (starting position).

```
byte myServo.moving();
```

Information about the state of motion of the servo.
0: The servos rests
> 0: Distance to go until the servo reaches its target position. In% of the total distance since the last 'write'.

```
byte myServo.read();
```

Current position of the servos in degrees.

```
byte myServo.readMicroseconds();
```

Current position of the servo in microseconds (pulse length).
In contrast to standard Lib these values are not necessarily the values passed with the last 'write' call. While the servo is still moving, the actual position is returned.

Although there is no separate 'stop' command, this can be accomplished with the sequence
`myServo.write( myServo.readMicroseconds() );`
Thus the current position is set as the new target position, which leads to the immediate stop of the servo.

```
byte myServo.attached();
```

Returns 'true' if a pin is assigned.

```
void setMinimumPulse( word length);
```

Defines the pulse length for angle '0'

```
void setMaximumPulse( word length);
```

Defines the pulse length for angle '180'

Class Stepper4

Unipolar stepper motors can be controlled thereby. In contrast to the Arduino standard library, the calls are nonblocking. The program in the 'loop' continues to run while the stepper motor is turning.

Setting up of the stepper motor:

```
Stepper4 myStepper( int steps360, byte mode  );
```

mode specifies whether the engine is activated in `FULLSTEP` or `HALFSTEP` mode. If the parameter is omitted, `FULLSTEP` is assumed. `steps360` is the number of steps the motor takes for one revolution (in the specified Mode)

Functions:

```
byte myStepper.attach( byte spi );
byte myStepper.attach( byte pin1, byte pin2, byte pin3, byte pin4 );
```

Assignment of the output signals. The signals can be assigned either to 4 individual pins, or to the SPI interface. For the SPI interface the values SPI_1, SPI_2, SPI_3 or SPI_4 are permitted. There are always 16 bit shifted out. The parameters indicate the position of the 4 bits of the assigned motor. SPI_4 are the first pushed out bits and are therefore at the end of the shift register chain. If only a 8-bit shift register connected, only the values of SPI_1 and SPI_2 can be used.
The SPI interface is enabled only if at least one of the stepper motor is connected via SPI. An example of the wiring is shown below.

The function value is 0 ('false') if no mapping was possible.

```
void myStepper.detach();
```

The pin assignment is canceled.

```
void myStepper.setSpeed(int rpm10 );
```

Set up rotation speed (in revolutions / min). The value must be specified in rpm*10 .

```
void myStepper.doSteps(long stepcount );
```

Number of steps to be performed by the engine. The sign indicates the direction of rotation.

```
void myStepper.rotate( int direction );
```

The motor rotates up to the next stop command.

```
void myStepper.stop( );
```

Stops the motor immediately.

```
void myStepper.setZero( );
```

The current position of the motor is taken as a reference point for the 'write commands with absolute positioning.

```
void myStepper.write( long angle );
void myStepper.write( long angle, byte factor );
```

The motor moves (measured from SetZero-point) to the specified angle. The passed angle is not limited to 360 °. For example setting angel = 3600 means 10 revolutions from SetZero point. If, for example, next call angel = -360 is passed, this does not mean 1 turn back, but 11 turns back! (-360 ° from SetZero point)
The 2nd call allows to specify the angle as a fraction. With factor = 10 set angle is interpreted in 1/10 °.

```
void myStepper.writeSteps( long stepPos );
```

The motor will move to the position that is stepPos steps away from SetZero point

```
byte myStepper.moving();
```

Information about the state of motion of the stepper.
= 0 when the engine stopped
= 255 when the motor rotates endlessly ('rotate' call)
> 0 ... <= 100 rest of movement to the destination point in% of total distance since last 'write' or 'doSteps' command.

```
long myStepper.read();
```
returns to actual position of the stepper in degrees (measured from SetZero point).

```
long myStepper.readSteps();
```
returns to actual position of the stepper in steps (measured from SetZero point).

Absolute and relative commands can be used mixed. Internally, the steps taken are always counted (in a long variable). As long as this variable does not overflow (happens after about two billion steps in one direction), the controller always knows where the stepper is. 'SetZero' sets this counter to 0.

Klasse SoftLed

The class SoftLED contains methods to switch a LED on and off softly.

Create:
```
SoftLed myLed;
```

Functions:
```
byte myLed.attach( byte pinNo );
```
The pin must have PWM capability, that is, it must be possible to do an analogWrite() to this pin. The pins 9 + 10 cannot be used.

```
void myLed.riseTime( int value );
```
The value indicates the time in ms until the Led reaches full brightness or darkness.

```
void myLed.on();
```
The LED is turned on.

```
void myLed.off();
```
The LED is turned off.

Klasse Eggtimer

This class contains methods for non blocking time delays.

Set up:
```
EggTimer myTimer
```

Functions:
```
void myTimer( long time );
```
The value specifies the duration of the delay in ms.

```
bool myTimer.running();
```
'true' while the timer is running, false otherwise .

Appendix:
Example diagram how to connect a step motor via the SPI interface: