# MobaTools – eine Arduino Library für die Modellbahn

Dokumentation zu Version 1.1

Die MobaTools sind eine Zusammenstellung von Funktionen, die speziell im Modellbahnumfeld ( aber nicht nur dort ) oft gebraucht werden können. Sie erleichtern einige Standard-Aufgaben und ermöglichen so auch Einsteigern komplexe Sketches zu schreiben.

Zusammenfassung der Klassen:

| | |
|---|---|
| Servo8 | Ansteuerung von bis zu 16 Servos an beliebigen Pins. Die Aufrufe sind weitgehend kompatibel zur Standard Servolib des Arduino. Allerdings kann die Geschwindigkeit, mit der sich das Servo bewegt, vorgegeben werden. |
| Stepper4 | Ansteuerung von Schrittmotoren. Im Gegensatz zur Standard Library 'Stepper' sind die Aufrufe nicht blockierend. Nachdem ein Referenzpunkt festgelegt wurde, lässt sich der Schrittmotor mit nahezu den gleichen Aufrufen wie in der Servo8 – Library auch absolut positionieren (in Winkelgraden)<br>Ab V0.7 wird der Schrittmotortreiber A4988 unterstützt. Damit können bipolare Schrittmotoren (auch mit Microstepping) angesteuert werden.<br>Ab V1.1 können auch Anfahr- und Bremsrampen definiert werden |
| SoftLed | Weiches auf- und Abblenden von Leds. Als Anschlußpins sind alle Digitalpins zulässig,. |
| EggTimer | 'Kurzzeitwecker' der einfach Zeitverzögerungen ohne Blockierung des Sketchablaufes im loop ermöglicht. |

Die MobaTools verwenden intensiv den Timer1 des Arduino. D.h. Alle anderen Libraries oder Funktionen, die ebenfalls den Timer1 verwenden, können nicht gleichzeitig genutzt werden. Dies betrifft z.B. die Standard-ServoLib und analogWrite an den Pins 9+10.
Ab der Version 1.0 wird der Timer 3 verwendet, wenn er vorhanden ist. Das betrifft Arduino Leonardo, Micro und Mega.

Obwohl es für Softleds und Stepper keine harte Grenze bezüglich der Zahl der instanziierbaren Objekte mehr gibt, sind die Leistungsgrenzen des verwendeten Prozessors zu beachten. Sowohl Stepper als auch Softleds werden in ein und derselben Interruptroutine verwaltet. Softleds belasten den Interrupt nur während des Auf- und Abblendens. Bei Steppern ist die Belastung abhängig von der Steprate (Häufigkeit des Interrupts) und der Rampe ( Während der Rampe ist die Rechenbelastung im Interrupt höher ).
Bei UNO/Nano/Leonardo und Micro ergibt sich die Begrenzung bereits aus der Zahl der verfügbaren Pins. Bei einem Mega ist aber verstärkt darauf zu achten.
Versuche mit 6 Steppern und 6 ständig pulsierenden Leds waren noch problemlos.

Seit der Version V.09 sind die MobaTools auch auf der STM32F1 Plattform ‚stm32duino‘ lauffähig ( z.B. MapleMini )

## Klasse Servo8

Die Servo Funktionen erzeugen die Impulse verschachtelt innerhalb eines 20ms Zyklus. Da die 20ms fest sind, müssen innerhalb dieser Zeit alle Impulse erzeugbar sein. Die Zahl der erzeugbaren

Impulse hängt von der Maximallänge und – da die Impulse überlappend erzeugt werden – auch von der Minimallänge der Impulse ab. Standardmäßig liegen die Grenzen bei 0,7ms (Min) und 2.3ms maximal. Daraus ergibt sich die Zahl von 16 anschließbaren Servos.

Einrichten des Servos:

```
Servo8     myServo;
```

Funktionen:

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

Ordnet dem Servo einen pin zu, an dem die Impulse ausgegeben werden. Der Pin wird auf Ausgang geschaltet, es werden aber noch keine Impulse erzeugt.

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

Wird der optionale Parameter 'autoOff' mit dem Wert 'true' übergeben, so wird der Impuls automatisch abgeschaltet, wenn sich die Länge für mehr als 1sec nicht ändert.

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

Mit den Parametern pos0 und pos180 kann vorgegeben werden, welche Impulslängen den Winkeln '0' bzw. '180' zugeordnet werden. ( der Parameter autoOff kann auch hier entfallen)

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

Die Zuordnung des Servos zum Pin wird aufgehoben, es werden keine Impulse mehr erzeugt.

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

Bewegungsgeschwindigkeit des Servos vorgeben. 'Speed' ist der Wert ( in 0,5µs Einheiten) um den sich die Impulslänge alle 20ms verändert, wenn sich das Servo bewegt. Speed=1 bedeutet, dass 40s gebraucht werden, um die Impulslänge von 1ms auf 2ms zu ändern. Speed=0 ( default Wert) bedeutet direkte Impulsänderung (wie Standard-Servo Library )

Ab Version 0.9 ist Auflösung des Speed-Parameters 4x größer. D.h. der Wert ist in 0,125µs Einheiten aufgelöst. Aus Kompatibilitätsgründen startet die Lib in einem Kompatibilitätsmode, der sich verhält wie vorher. Dieser Modus kann mit einem zusätzlichen Parameter global, d.h. für alle eingerichteten Servos, abgeschaltet werden:

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

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

Zielposition vorgeben. Der Servo bewegt sich mit der vorgegebenen Geschwindigkeit zu dieser Position. Bei Werten von 0...180 Wird der Wert als Winkel interpretiert, und entsprechend in eine Impulslänge umgerechnet. Werte > 180 werden als Impulslänge in µs interpretiert, wobei auf die minimale (700µs) bzw. maximale (2300µs) Impulslänge begrenzt wird.

Der write-Befehl wirkt sofort, auch wenn das Servo seine beim vorhergehenden write gesetzte Position noch nicht erreicht hat. Gegebenenfalls ändert das Servo auch sofort die Drehrichtung.

Ist dies der erste 'write' Aufruf nach einem 'attach', so wird die vorgegebene Impulslänge sofort ausgegeben (Startposition).

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

Information über den Bewegungszustand des Servo.

0: Das Servos steht still

>0: Restweg, den das Servo bis zum Ziel noch hat. In % vom Gesamtweg seit dem letzten 'write'.

`byte myServo.read();`
Momentane Position des Servos in Winkelgraden.
`byte myServo.readMicroseconds();`
Momentane Position des Servos in µs (Impulslänge).
Im Gegensatz zur Standard-Lib entsprechen diese Werte nicht unbedingt dem mit dem letzten 'write' übergebenen Wert. Während sich das Servo noch bewegt, wird die tatsächliche Position zurückgegeben.
Obwohl es keinen eigenen 'stop' Befehl gibt, lässt sich dies mit der Sequenz
`myServo.write( myServo.readMicroseconds() );`
erreichen. Damit wird die Momentanposition als neue Zielposition vorgegeben, was zum sofortigen Stop des Servos führt.

`byte myServo.attached();`
Gibt 'true' zurück, wenn ein Pin zugeordnet ist

`void setMinimumPulse( word length);`
Ordnet dem Winkel '0' eine Impulslänge zu
`void setMaximumPulse( word length);`
Ordnet dem Winkel '180' eine Impulslänge zu

## Klasse Stepper4

Damit können bipolare und unipolare Schrittmotore angesteuert werden. Im Gegensatz zur Arduino Standard Library sind die Aufrufe nicht blockierend. Das Programm im 'loop' läuft weiter, während sich der Schrittmotor dreht.
Neben der Schrittgeschwindigkeit kann auch einen Anfahr- und Bremsrampe vorgegeben werden. Die Rampenlänge wird über die Schrittzahl definiert. D.h. es wird angegeben, wie viele Schritte benötigt werden, um vom Stillstand auf die eingestellte Geschwindigkeit zu kommen.
Bei Rampenlänge 0 (default) verhält sich der Schrittmotor wie bisher ( vor V1.1 ).
Wird die Schrittgeschwindigkeit verändert ohne eine neue Rampenlänge vorzugeben, so wird die Rampenlänge so angepasst, dass das Anfahrverhalten etwa gleich bleibt.

Einrichten des Schrittmotors:
`Stepper4 myStepper( int steps360, byte mode  );`
`mode` gibt an, ob der Motor im `FULLSTEP` oder `HALFSTEP` Modus angesteuert wird. Wird der Parameter weggelassen, wird `HALFSTEP` angenommen.
mode=`A4988` gibt an, dass der Motor über einen Schrittmotortreiber mit Step- und Direction Eingang ( z.B. der A4988 ) angeschlossen ist.

`steps360` ist die Zahl der Schritte, die der Motor für eine Umdrehung braucht ( im jeweils angegebenen Mode, bei `A4988` ist das eingestellte Microstepping zu berücksichtigen)

Funktionen:
`byte myStepper.attach( byte spi );`
`byte myStepper.attach( byte pin1, byte pin2, byte pin3, byte pin4 );`
Zuordnung der Ausgangssignale. Die Signale könen entweder an 4 einzelnen pins, oder er die SPI-Schnittstelle ausgegeben werden. Für den Parameter `spi` sind die Werte `SPI_1, SPI_2, SPI_3 oder SPI_4` zulässig. Es werden immer 16 Bit herausgeschoben. Die Parameter geben an, an welcher Position die 4er Gruppe steht. `SPI_4` sind die zuerst herausgeschobenen Bits und stehen deshalb am Ende der Schieberegisterkette.

Wird nur ein 8-Bit Schieberegister angeschlossen, können nur die Werte `SPI_1` und `SPI_2` genutzt werden.
Die SPI-Schnittstelle wird nur aktiviert, wenn mindestens ein Schrittmotor über SPI angeschlossen wird. Ein Beispiel zur Beschaltung ist im Anhang zu sehen.

Funktionswert ist 0 ('false') wenn keine Zuordnung möglich war.
Über die 4 Ausgangspins können sowohl unipolare als auch ( über eine doppelte H-Brücke ) bipolare Motoren angesteuert werden.

```
byte myStepper.attach( byte pinStep, byte pinDir );
```
Bei der Ansteuerung über den A4988 ( oder kompatiblen ) Treiber werden hier die Ports für ‚Step' und ‚Direction' angegeben.
Funktionswert ist 0 ('false') wenn keine Zuordnung möglich war.

```
void myStepper.detach();
```
Die Pinzuordnung wird aufgehoben.

```
uint16_t myStepper.setSpeed(int rpm10 );
```
Rotationsgeschwindigkeit ( in Umdrehungen / min ) einstellen. Der Wert muss in rpm*10 angegeben werden.
Funktionswert ist die aktuelle Rampenlänge

```
uint16_t myStepper.setSpeedSteps( uint16_t speed10 );
```
Schrittgeschwindigkeit des Motors in Steps/10sec. Maximalwert ist 12500 ( 1250Steps/sec). Die Rampenlänge wird gegebenenfalls angepasst, um das Anfahrverhalten in etwa gleich zuhalten.
Funktionswert ist die aktuelle Rampenlänge.

```
uint16_t myStepper.setSpeedSteps( uint16_t speed10, uint16_t rampLen );
```
Schrittgeschwindigkeit des Motors in Steps/10sec.
Rampenlänge in Steps.
Die zulässige Rampenlänge ist von der Schrittgeschwindigkeit abhängig, und maximal 16000 für hohe Stepraten. Bei Stepraten unter 2Steps/sec ist keine Rampe mehr möglich. Liegt ramplen außerhalb des zulässigen Bereiches, wird der Wert angepasst.
Funktionswert ist die aktuelle (gegebenenfalls angepasste) Rampenlänge.

```
uint16_t myStepper.setRampLen( uint16_t rampLen );
```
Rampenlänge in Steps. Die zulässige Rampenlänge ist von der Schrittgeschwindigkeit abhängig, und maximal 16000 für hohe Stepraten. Bei Stepraten unter 2Steps/sec ist keine Rampe mehr möglich. Liegt ramplen außerhalb des zulässigen Bereiches, wird der Wert angepasst.
Funktionswert ist die aktuelle (gegebenenfalls angepasste) Rampenlänge.

```
void myStepper.doSteps(long stepcount );
```
Zahl der Schritte, die der Motor ausführen soll. Das Vorzeichen gibt die Drehrichtung an. Referenzpunkt für den Start der Schritte ist immer die momentane Motorposition. Und zwar auch dann, wenn sich der Motor zum Zeitpunkt des Befehls bereits dreht. Es wird dabei aus der Zahl der Schritte eine neue Zielposition errechnet, die dann angefahren wird ( mit Rampe ). Dies kann dazu führen, dass der Motor erst über das Ziel hinausfährt, und sich dann zurück zur Zielposition dreht.
doSteps(0) führt z.B. dazu, dass der Motor abbremst und zu der Position zum Zeitpunkt des Befehls zurückdreht,

```
void myStepper.rotate( int direction );
```
Der Motor dreht bis zum nächsten Stop-Befehl.( 1= forwarts, -1 = rückwarts )
Rotate(0) hält den Motor an (mit Rampe). Der Motor bleibt am Ende der Bremsrampe stehen.

```
void myStepper.stop( );
```
Hält den Motor sofort an (Notstop).

```
void myStepper.setZero( );
```
Die aktuelle Position des Motors wird als Referenzpunkt für den 'write Befehl mit absoluter Positionierung genommen.

```
void myStepper.write( long angle );
void myStepper.write( long angle, byte factor );
```
Der Motor bewegt sich zum angegebenen Winkel ( gemessen von setZero-Punkt ). Der vorgebbare Winkel ist nicht auf 360° beschrankt. Z.B. bedeutet angel=3600 10 Umdrehungen vom setZero Punkt. Wird danach z.B. angel= -360 übergeben, bedeutet das nicht 1 Umdrehung zurück, sondern 11 Umdrehungen zurück! (-360° vom setZero Punkt)

Der 2. Aufruf erlaubt es, den Winkel als Bruch anzugeben. Wird factor = 10 gesetzt, wird angle in 1/10° interpretiert.

```
void myStepper.writeSteps( long stepPos );
```
Der Motor bewegt sich zu der Position, die stepPos Schritte vom setZero Punkt entfernt ist

```
byte myStepper.moving();
```
=0 wenn der Motor steht
>0...<=100 Restlicher Weg bis zum Zielpunkt in % vom Gesamtweg seit letztem 'write' oder 'doSteps' Befehl. Dies beinhaltet auch gegebenenfalls einen ‚Umweg', wenn aufgrund der Rampe der Stepper erst über das Ziel hinaus, und dann wieder zurückbewegt wird. In diesem Fall können Werte über 100% entstehen.

```
long myStepper.stepsToDo();
```
gibt zurück wie viel Schritte noch bis zum Ziel ausgeführt werden müssen. Dies beinhaltet auch gegebenenfalls einen ‚Umweg', wenn aufgrund der Rampe der Stepper erst über das Ziel hinaus, und dann wieder zurückbewegt wird.

```
long myStepper.read();
```
gibt zurück an welchem Winkel sich der Motor befindet ( gemessen vom setZero Punkt).

```
long myStepper.readSteps();
```
gibt zurück wie viel Schritte sich der Motor vom setZero Punkt entfernt befindet.

Absolute und relative Befehle können gemischt verwendet werden. Intern werden die ausgeführten Schritte immer mitgezählt ( in einer long Variablen ). Solange diese Variable nicht überläuft ( passiert nach gut 2Mrd Steps in eine Richtung ), weiß die Steuerung immer, wo sich der Motor befindet. 'setZero' setzt diesen Zähler zu 0.

## Klasse SoftLed

Die Klasse SoftLed enthält Methoden, um eine Led 'weich' ein- und auszuschalten. Die PWM-Impulse während des Auf/Abblendens werden per Software erzeugt. Daher funktioniert dies an allen digitalen Ausgängen.

Einrichten:
```
SoftLed myLed;
```
Die Zahl der ansteuerbaren Leds ist grundsätzlich nicht begrenzt. In der Praxis bestimmen die vorhanden Pins und die Leistungsfähigkeit des Microprozessors (Ram und Geschwindigkeit) die sinnvoll nutzbare Anzahl.

Funktionen:
```
byte myLed.attach( byte pinNr );
byte myLed.attach( byte pinNr, byte Invert );
```
Ab der Version 08 der MobaTools können alle Digitalausgänge verwendet werden. Ist der Parameter Invert angegeben und ‚True', so ändert sich die Ausgangslogik: ON ist dann LOW am Ausgang, OFF ist HIGH am Ausgang.

```
void myLed.riseTime( int Wert );
```
Der Wert gibt die Zeit in ms an, bis die Led ihre volle Helligkeit erreicht hat bzw. dunkel ist. Da intern nur bestimmte Stufen möglich sind, ist die tatsächliche Zeit eine möglichst gute Annäherung an den übergebenen Wert.
Der Maximalwert sind 1280 ms.

```
void myLed.on();
```
Die Led wird eingeschaltet.

```
void myLed.off();
```
Die Led wird ausgeschaltet.

```
void myLed.toggle();
```
Die Led wird umgeschaltet.

```
void myLed.write( byte Zustand);
```
Zustand = ON oder OFF. Die Led wird entsprechend ‚Zustand' ein- oder ausgeschaltet.

```
void myLed.write(byte Zustand, byte Type );
```
mit Type = LINEAR oder BULB kann die Charakteristik des Auf- Abblendens verändert werden.

## Klasse Eggtimer

Die Klasse EggTimer enthält Methoden für einfache Zeitverzögerungen.

Einrichten:
```
EggTimer myTimer
```

Funktionen:
```
void myTimer.setTime( long Zeit );
```
Startet den Timer. Der Wert gibt die Laufzeit des Timers in ms an.
Maximalwert sind 2 Mrd ms

```
bool myTimer.running();
```
'true' während der Timer läuft, 'false' sonst..

```
long myTimer.getTime();
```

gibt die Restlaufzeit des Timers zurück ( in ms ).

Anhang:

Beispielschema des Anschlusses eines Schrittmotors über die SPI-Schnittstelle: