# AccelStepper 1.58

*Knihovna Arduina umožňuje prostřednictvím svých tříd AccelStepper a MultiStepper objektový přístup k řízení krokových motorů.*

## Základní vlastnosti:

- Řízení krokových motorů v režimu driveru se signály STEP a DIR nebo přímým spínáním fází dvou, tří nebo čtyřkanálovými drivery.
- Nastavení automatického zrychlování a zpomalování (akcelerace a decelerace).
- Absolutní i relativní polohování.
- Řízení s konstantní rychlostí otáčení.
- Současné řízení více krokových motorů s různým zrychlením a rychlostí otáčení.
- Sdružení motorů do skupin (třída MultiStepper), ve kterých budou rychlosti otáčení jednotlivých motorů synchronizovány tak, aby všechny motory dosáhly své cílové pozice současně.

### Poznámka:

*Knihovna je realizována jako pollovací[1], výkonné metody vytvořených objektů je tedy nutné volat v hlavní smyčce programu s dostatečnou frekvencí.*

*Některé úlohy jsou obslouženy pomocí blokujících volání, tj. program v těle funkce setrvá tak dlouho, dokud není úloha zcela splněna.*

*Neexistuje žádná zpětná vazba mezi skutečnou pozicí motoru a pozicí vypočtenou z výchozí pozice a počtu odeslaných kroků (řízení s otevřenou regulační smyčkou). Pokud se krokový motor násilně zastaví nebo dojde k překročení jeho maximální rychlosti otáčení a tím ke ztrátě kroků, AccelStepper to nezaregistruje a dojde k chybné funkci zařízení.*

[1] *Jednoduchý princip, postavený na přímé komunikaci, jehož zásadní nevýhodou je, že vyžaduje pro provádění akcí i pro čtení dat opakovanou aktivitu*

## Způsob práce

Pro pohyb motoru počítá výkonná funkce `run()` objektu rychlost (délku prodlevy mezi impulzy) krokování v mikrosekundách. Rychlost krokování je přepočítána po každém kroku a parametry rychlosti a zrychlení je tak možno měnit před každým voláním funkce `run()`. Při každém volání funkce `run()` vykoná motor buď jeden, nebo žádný krok v závislosti na tom, zda je v daný okamžik potřebné krok vykonat, či nikoliv.
Funkce `run()` musí být opakovaně a dostatečně rychle volána tak dlouho, dokud motor nedosáhne požadované polohy; po dosažení polohy je již funkce `run()` nečinná.

### Pozicování:

Po vytvoření instance třídy AccelStepper (tj. vytvoření jednoho objektu typu AccelStepper) je aktuální pozice motoru nastavena na nulu, ale je ji možné změnit funkcí `currentPosition()`.
Hodnota určující okamžitou pozici se zvyšuje při otáčení motoru v jednom směru; při otáčení opačným směrem se hodnota pozice snižuje a případně přechází do záporných hodnot.
Pozice jsou ukládány ve formátu *signed long integer*[2].

### Výkonnost:

Při taktovací frekvenci Arduina 16 MHz je nejvyšší spolehlivě dosažitelná rychlost otáčení přibližně 4 000 kroků za sekundu za předpokladu, že je funkce `run()` volána tak často, aby se motor mohl pootočit o další krok, kdykoliv je to třeba. Rychlejší procesory umožní větší rychlosti krokování.
Knihovna dovoluje používat i velmi nízké rychlosti krokování (mnohem pomalejší než jeden krok za sekundu).
Výkonnost výrazně ovlivňuje frekvence volání funkce `setAcceleration()`, které je časově náročné, protože vyžaduje výpočet odmocniny.

[2] *Dlouhý integer (4 byty) se znaménkem*

# *Třída AccelStepper*

*Tato třída umožňuje pokročilé řízení jednoho krokového motoru. Objektů, vzniklých instanciací této třídy, může být použito v programu více a tedy je možné řídit více různých krokových motorů najednou. Jejich maximální počet není možné obecně určit, neboť záleží na velikostí paměti použitého Arduina, jeho rychlostí a velikostí a požadavcích konkrétního programu.*
*Pro synchronizovaný pohyb krokových motorů se používá třída MultiStepper, která je součástí knihovny AccelStepper.*

## Konstruktor[3]

## AccelStepper()

*Konstruktor vytvoří pojmenovanou instanci třídy AccelStepper().*

Vytvoříte-li více instancí, můžete – pokud ovšem voláte funkci `run()` dostatečně často – současně pohybovat několika krokovými motory různou rychlostí, s různým zrychlením a různým směrem.

### Syntaxe:

```
AccelStepper stepper1(interface, pin1, pin2, pin3, pin4,
[enable]);
```

V následujících příkladech bude jako jméno instance používán symbolický název **stepper1**

### Parametry:

**interface** (*byte*) – volba způsobu ovládání motoru.
Jako parametr můžete použít buď čísla **1**, **2**, **3**, **4**, **6** a **8**, nebo názornější symbolické názvy **MotorInterfaceType** (*viz příloha*). Výchozí nastavení je **4** (**FULL4WIRE**).
**pin1** (*byte*) – první výstup pro motor.
Ve výchozím stavu je mu přiřazen fyzický pin **D2**. Pro režim **1** (**DRIVER**) je to signál **STEP**. Aktivní je náběžná hrana signálu.
**pin2** (*byte*) – druhý výstup pro motor.
Ve výchozím stavu je mu přiřazen fyzický pin **D3**. Pro režim **1** (**DRIVER**), to je signál **DIR**.
**pin3** (*byte*) – třetí výstup pro motor. Ve výchozím stavu je mu přiřazen fyzický pin **D4**.

[3] *Konstruktor je speciální funkce (metoda) třídy, která se volá ve chvíli vytváření instance této třídy*

**pin4** (*byte*) – čtvrtý výstup pro motor. Ve výchozím stavu je mu přiřazen fyzický pin **D5**.
**enable** (*bool*) – [*nepovinné*] pokud je hodnota **TRUE** (výchozí stav), bude během inicializace knihovny voláním funkce *enableOutputs()* nastaven i tento pin jako výstupní.

### Poznámka:

*Po vytvoření instance je aktuální pozice motoru nastavena na hodnotu 0, cílová pozice je nastavena na 0, maximální rychlost na 1,0 a zrychlení na 1,0.*

### Příklad:

Konstruktor ve formátu:

```
AccelStepper stepper1(1, 2, 3);
```

vytvoří instanci třídy AccelStepper, pojmenovanou *stepper1* v konfiguraci pro driver, řízený signály **STEP** (pin 2) a **DIR** (pin 3).

Konstruktor ve formátu:

```
AccelStepper stepper2(4, 2, 3, 4, 5);
```

vytvoří instanci třídy AccelStepper, pojmenovanou *stepper2* v konfiguraci pro přímé řízení dvoufázového unipolárního krokového motoru s výkonovými spínači připojenými na piny 2 až 5.

### Poznámka:

*Příklady použití symbolických názvů MotorInterfaceType naleznete v příloze.*

# *Členské funkce (metody)*[4]

## Konfigurační funkce

### setEnablePin()

*Povoluje použití hardwarového blokování funkce driveru.*

#### Syntaxe:

```
stepper1.setEnablePin(enablePin);
```

#### Parametr:

**enablePin** (*uint8_t*) – parametr určuje, který pin bude ovládat vstup ENABLE driveru krokového motoru.
Pokud je jako parametr funkce použita hodnota 255 (výchozí nastavení) znamená to, že vstup ENABLE není u driveru použit.
Funkční jsou hodnoty 0 až 13 a 255.

#### Návratová hodnota:

Funkce nic nevrací.

#### Poznámka:

*Pokud je určenému pinu nastavena funkce ENABLE, pin se nastaví do aktivní úrovně, když je volána funkce* `enableOutputs()` *a nastaví do neaktivní úrovně, když je volána funkce* `disableOutputs()`.

### enableOutputs()

*Nastaví piny, určené pro ovládání motoru, jako výstupní.*

#### Syntaxe:

```
stepper1.enableOutputs();
```

[4] *Členská funkce = metoda (Member function)*

Je-li při výstavbě knihovny povoleno použití pinu ENABLE, funkce `enableOutputs()` tento pin nastaví jako výstupní a zároveň ho nastaví do vysoké (log.1, HIGH) logické úrovně. Při inicializaci knihovny je tato funkce automaticky volána konstruktorem.

### Parametry:

Funkce nemá žádné parametry.

### Návratová hodnota:

Funkce nic nevrací.

## disableOutputs()

*Odpojuje motor od napájení nastavením všech řídících výstupů na nízkou (log. 0, LOW) logickou úroveň.*

### Syntaxe:

```
stepper1.disableOutputs();
```

### Parametry:

Funkce nemá žádné parametry.

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*Je-li při výstavbě knihovny povoleno také použití pinu ENABLE, je voláním této funkce zároveň nastaven do nízké (log.0, low) logické úrovně. Funkce je užitečná především při uvádění Arduina do režimu nízké spotřeby, její skutečná funkčnost ovšem závisí na konkrétním provedení driveru motoru. Před zahájením další činnosti je nutno řídící výstupy znovu aktivovat funkcí* `enableOutputs()`.

## setMinPulseWidth()

*Nastaví minimální dobu trvání výstupního impulsu pro driver krokového motoru.*

### Syntaxe:

```
stepper1.setMinPulseWidth(width);
```

### Parametr:

**width** (*byte)* – funkční jsou hodnoty 20 až 255.
Nejmenší prakticky použitelná doba trvání impulsu je přibližně 20 mikrosekund.

### Návratová hodnota:

Funkce nic nevrací.

## setPinsInverted()

*Invertuje polaritu signálů pro řízení motoru.*

### Syntaxe pro driver řízený signály STEP, DIR a ENABLE:

```
stepper1.setPinsInverted(directionInvert, stepInvert,
enableInvert);
```

### Parametry:

**directionInvert** (*uint8_t*) – **TRUE** znamená invertovaný výstup, **FALSE** normální
**stepInvert** (*uint8_t*) – **TRUE** znamená invertovaný výstup, **FALSE** normální
**enableInvert** (*uint8_t*) – **TRUE** znamená invertovaný výstup, **FALSE** normální

### Syntaxe pro přímé řízení fází motoru:

```
stepper1.setPinsInverted(pin1Invert, pin2Invert, pin3Invert,
pin4Invert, enableInvert);
```

### Parametry:

**pin1Invert** (*uint8_t*) – **TRUE** znamená invertovaný výstup, **FALSE** normální
**pin2Invert** (*uint8_t*) – **TRUE** znamená invertovaný výstup, **FALSE** normální
**pin3Invert** (*uint8_t*) – **TRUE** znamená invertovaný výstup, **FALSE** normální
**pin4Invert** (*uint8_t*) – **TRUE** znamená invertovaný výstup, **FALSE** normální
**enableInvert** (*uint8_t*) – **TRUE** znamená invertovaný výstup, **FALSE** normální

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*V popisu funkcí je pro hodnoty pinů použito označení vysoká / nízká úroveň (log.1, HIGH / log.0, LOW). V případě invertování polarity některých pinů pomocí této funkce se budou za*

*běhu programy piny fyzicky nastavovat naopak, tj. například povolení funkčnosti (aktivace) řadiče motorů pomocí pinu ENABLE se bude při invertování provádět nastavením tohoto pinu na nízkou úroveň (log.0, LOW) a jeho deaktivace nastavením na úroveň vysokou (log.1, HIGH).*

## Nastavení parametrů

### move()

*Nastavuje cílovou pozici relativně vzhledem k aktuální pozici.*

#### Syntaxe:

```
stepper1.move(relative);
```

#### Parametr:

**relative** (*signed long integer*) – požadovaná poloha vzhledem k aktuální pozici.
Znaménko parametru určuje směr otáčení motoru.

#### Návratová hodnota:

Funkce nic nevrací.

#### Poznámka:

*Funkce* `move()` *nastaví pozici pomocí funkce* `moveTo()`*. Vztahuje se na ní proto také poznámka uvedená u* `moveTo()`*; pokud je požadován pohyb konstantní rychlostí, je třeba před voláním* `move()` *volat i funkci* `setSpeed()`.

### moveTo()

*Nastavuje cílovou polohu.*

#### Syntaxe:

```
stepper1.moveTo(absolute);
```

#### Parametr:

**absolute** (*signed long integer*) – požadovaná absolutní poloha.
Znaménko parametru určuje směr otáčení motoru.

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*Funkce* `run()` *při každém volání zjišťuje, zda je již nutno pootočit motorem o jeden krok ze současné pozice směrem k cílové pozici, zadané funkcí* `moveTo()`*. Funkce* `moveTo()` *zároveň přepočítá rychlost pro další krok. Pokud chcete použít pohyb konstantní rychlostí, volejte po změně cílové pozice (relativní nebo absolutní) funkcemi* `move()` *nebo* `moveTo()` *ještě funkci* `setSpeed()`.

## setAcceleration()

*Nastaví zrychlení / zpoždění (program zrychluje i zpomaluje stejně).*

### Syntaxe:

```
stepper1.setAcceleration(acceleration);
```

### Parametr:

**acceleration** (*float*) – požadované zrychlení v krocích za sekundu na druhou.
Musí být větší než 0,0.

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*Činnost funkce je časově náročná, protože vyžaduje výpočet odmocniny. Nevolejte ji proto častěji, než je nezbytně nutné.*

## setCurrentPosition()

*Nastaví aktuální pozici motoru jako výchozí a rychlost na 0,0.*

### Syntaxe:

```
stepper1.setCurrentPosition(position);
```

### Parametr:

**position** (*signed long integer*) – pozice v krocích.

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*Nastavení aktuální pozice motoru je užitečné například po hardwarovém nastavení výchozí (home) pozice. Vedlejším efektem je nastavení aktuální rychlosti na hodnotu 0,0.*

## setMaxSpeed()

*Nastavuje maximální povolenou rychlost otáčení motoru.*

### Syntaxe:

```
stepper1.setMaxSpeed(speed);
```

### Parametr:

**speed** (*float*) – požadovaná maximální rychlost v krocích za sekundu.
Musí být větší než 0.

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*Funkce* `run()` *bude zrychlovat jen do rychlosti, nastavené funkcí* `setMaxSpeed()`. *Maximální dosažitelná rychlost záleží na výkonnosti procesoru a jeho taktovací frekvenci; nastavení vyšší rychlosti může vést k nelineárnímu průběhu zrychlování a zpomalování.*

## setSpeed()

*Nastavuje velikost konstantní rychlosti otáčení motoru při použití funkce* `runSpeed()`.

### Syntaxe:

```
stepper1.setSpeed(speed);
```

### Parametr:

**speed** (*float*) – požadovaná konstantní rychlost v krocích za sekundu.

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*Při nastavení rychlosti větší než 1000 kroků za sekundu se už motor nemusí spolehlivě rozběhnout. Je ale možno nastavit velmi malé rychlosti otáčení (například hodnota 0,00027777 generuje přibližně jeden krok za hodinu). Přesnost nastavení rychlosti závisí na přesnosti oscilátoru Arduina. Fázový šum (jitter) závisí na tom, jak často budete volat funkci* `runSpeed()`.

## Výkonné metody

## run()

*Funkce při každém volání otočí motorem o jeden krok, pokud je to nutné k dosažení cílové pozice; přitom bere v úvahu nastavené zrychlení a zpomalení.*

### Syntaxe:

```
stepper1.run();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota (*bool*)

**TRUE**, pokud motor vykonal krok

**FALSE** v opačném případě

### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
bool pohyb;
pohyb = stepper1.run();
```

### Poznámka:

*Funkci je třeba volat co nejčastěji, nejméně však jedenkrát za minimální periodu krokování, nejlépe v hlavní smyčce programu.*

## runToPosition()

*Otáčí motorem (se zrychlením i zpomalením) až do dosažení aktuálně nastavené cílové pozice.*

### Syntaxe:

```
stepper1.runToPosition();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*Nepoužívejte ve smyčce, která zpracovává událostí, protože funkce blokuje další provádění programu až do doby dosažení nové cílové pozice.*

## runToNewPosition()

*Otáčí motorem (se zrychlením i zpomalením) až do dosažení nové cílové pozice.*

### Syntaxe:

```
stepper1.runToNewPosition(position);
```

### Parametr:

**position** (*signed long integer*) – nová cílová pozice.

### Návratová hodnota:

Funkce nic nevrací.

### Poznámka:

*Nepoužívejte ve smyčce, která zpracovává událostí, protože funkce blokuje další provádění programu až do doby dosažení nové cílové pozice.*

## runSpeed()

*Funkce zajišťuje otáčení konstantní rychlostí; pokud je to vzhledem k nastavené rychlosti potřeba, otočí motorem o jeden krok. Rychlost krokování je konstantní, nastavená funkcí* `setSpeed()`.

### Syntaxe:

```
stepper1.runSpeed();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota (*bool*):

**TRUE**, pokud motor vykonal krok

**FALSE** v opačném případě

### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
bool pohyb;
pohyb = stepper1.runSpeed();
```

### Poznámka:

*Funkci* `runSpeed()` *je třeba volat co nejčastěji, nejméně však jedenkrát za periodu krokování.*

## runSpeedToPosition()

*Otáčí motorem konstantní rychlostí, nastavenou funkcí* `setSpeed()`*, až do okamžiku dosažení cílové polohy. Nepoužívá zrychlení.*

### Syntaxe:

```
stepper1.runSpeedToPosition();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota:

**TRUE**, pokud se motor stále otáčí

**FALSE** v opačném případě

### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
bool pohyb;
pohyb = stepper1.runSpeedToPosition();
```

## stop()

*Vypočte nejbližší cílovou pozici, ve které se motor může zastavit bez ztráty kroku.*

**Syntaxe:**

```
stepper1.stop();
```

**Parametry:**

Funkce nemá žádné parametry

**Návratová hodnota:**

Funkce nic nevrací.

## Dotazy

### currentPosition()

*Dotaz na aktuální pozici motoru.*

**Syntaxe:**

```
stepper1.currentPosition();
```

**Parametry:**

Funkce nemá žádné parametry

**Návratová hodnota** (*signed long integer*)**:**

Vzdálenost v krocích od nulové pozice.
**Kladné číslo** určuje, o kolik kroků je motor vzdálen od nulové polohy v jednom směru
**Záporné číslo** určuje vzdálenost od nulové polohy v opačném směru.

**Příklad:**

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
signed long integer pozice;
pozice = stepper1.currentPosition();
```

### distanceToGo()

*Dotaz na zbývající počet kroků do cílové pozice.*

**Syntaxe:**

```
stepper1.distanceToGo();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota (*signed long integer*):

Zbývající počet kroků do cílové pozice

### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
signed long integer zbytek;
zbytek = stepper1.distanceToGo();
```

## isRunning()

*Zjišťuje, zda se zvolený motor pohybuje.*

### Syntaxe:

```
stepper1.isRunning();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota:

**TRUE**, pokud není rychlost nulová nebo pokud motor ještě nedosáhl cílové pozice.
**FALSE** v opačném případě.

### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
bool pohyb;
pohyb = stepper1.isRunning();
```

## maxSpeed()

*Dotaz na aktuálně platnou maximální rychlost, nastavenou funkcí* `setMaxSpeed()`.

### Syntaxe:

```
stepper1.maxSpeed();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota (*float*):

Aktuálně platná maximální rychlost.

### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
float rychlost;
rychlost = stepper1.maxSpeed();
```

## targetPosition()

*Dotaz na poslední nastavenou cílovou polohu.*

### Syntaxe:

```
stepper1.targetPosition();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota (*signed long integer*):

Cílová poloha v krocích.

### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
signed long integer cil;
cil = stepper1.targetPosition();
```

## speed()

*Dotaz na poslední nastavenou rychlost.*

### Syntaxe:

```
stepper1.speed();
```

### Parametry:

Funkce nemá žádné parametry

## **Návratová hodnota** (*float*)**:**

Poslední nastavená rychlost v krocích za sekundu.

## **Příklad:**

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
float rychlost;
rychlost = stepper1.speed();
```

# *Třída MultiStepper*

*Tato třída umožňuje sdružit více krokových motorů do jedné skupiny, aby po zadání požadované cílové pozice všechny jely synchronně tak, že na zvolenou pozici dojedou všechny zároveň. Jednotlivé motory se budou točit konstantní rychlostí (každý svou). Tato třída je tedy velmi užitečná například pro X-Y plottery, souřadnicové zapisovače nebo 3D tiskárny, kde je požadována lineární interpolace mezi libovolně určenými pozicemi v prostoru (2D, 3D i vyššími).*

### Poznámka:

*Třída pracuje pouze s konstantními rychlostmi; zrychlení a zpomalení (akcelerace a decelerace) nejsou podporovány. Všechny krokové motory, ovládané třídou MultiStepper, se budou pohybovat konstantní rychlostí (pro každý krokový motor jinou).*

## Konstruktor

## MultiStepper()

*Vytvoří instanci třídy Multistepper s názvem steppers.*

### Syntaxe:

```
MultiStepper steppers();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota:

Funkce nic nevrací.

### Příklad:

```
MultiStepper steppers();
```

## Členské funkce (metody)

### addStepper()

*Přidá krokový motor do skupiny. Maximální počet krokových motorů v jedné skupině je dán konstantou MULTISTEPPER_MAX_STEPPERS; ve standardní verzi knihovny je nastavena na 10.*

#### Syntaxe:

```
stepper1.addStepper(stepper);
```

#### Parametr:

**stepper** (*uint8_t*) – reference na krokový motor, který chcete přidat do této skupiny. Reference musí být objekt typu AccelStepper.

**Návratová hodnota** (*bool*)

**TRUE** – pokud se přidání zdařilo

**FALSE** – v případě, že by počet motorů ve skupině přesáhl maximum[5] a motor tedy nebylo možné do skupiny přidat[6].

#### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
bool uspech;
uspech = stepper1.addStepper();
```

### moveTo()

*Nastaví cílovou pozici všech motorů v souladu s předaným polem souřadnic. Nové rychlosti jednotlivých motorů budou spočítány tak, aby všechny motory do cílové pozice dospěly pokud možno ve stejnou dobu[7].*

#### Syntaxe:

```
steppers.moveTo(absolute[array]);
```

[5] *V případě nouze je možné na vlastní nebezpečí v hlavičkovém souboru třídy změnit konstantu MULTISTEPPER_MAX_STEPPERS.*

[6] *Neúspěch však tuto skupinu motorů nijak neovlivní a bude nadále fungovat jako dosud, pouze do ní tento motor nezačlení.*

[7] *Pokud by byla délka drah jednotlivých motorů příliš rozdílná, mohlo by se stát, že některý z motorů již cílové pozice dosáhne dříve, než jiný. Rozdíl ale bude v běžném případě nepatrný.*

### Parametr:

**absolute** (*array*) – pole požadovaných cílových pozic.
První položka pole (prvek s indexem 0) odpovídá motoru, který byl do skupiny přidán funkcí *addStepper()* jako první atd. Pole musí obsahovat alespoň tolik prvků, kolik je motorů ve skupině, jinak není chování definováno[8].

### Návratová hodnota:

Funkce nic nevrací.

## run()

*Zavolá funkci runSpeed() všech motorů ve skupině, které ještě nedosáhly cílové pozice.*

### Syntaxe:

```
steppers.run();
```

### Parametry:

Funkce nemá žádné parametry

### Návratová hodnota (*bool*):

**TRUE** – pokud se některý motor stále ještě pohybuje k cílové pozici.
**FALSE** – pokud již všechny motory dosáhly cílové pozice.

### Příklad:

```
// vytvoří proměnnou, do níž se ukládá návratová hodnota funkce
bool pohyb;
pohyb = steppers.run();
```

## runSpeedToPosition()

*Funkce otáčí všemi motory až do dosažení jejich cílových pozic.*

### Syntaxe:

```
steppers.runSpeedToPosition();
```

[8] *Pokud by bylo prvků v poli méně, způsobilo by to velmi pravděpodobně závažné potíže a motory by se pohybovaly nechtěným a velmi neočekávaným způsobem.*

## Parametry:

Funkce nemá žádné parametry

## Návratová hodnota:

Funkce nic nevrací.

## Poznámka:

*Nepoužívejte ve smyčce, která zpracovává událostí, protože funkce blokuje další provádění programu až do doby dosažení cílové pozice všech motorů. Pokud není blokování přípustné, použijte místo toho opakované volání funkce run().*

# *Příloha:*

## Symbolické názvy MotorInterfaceType

**AccelStepper::DRIVER** (*= 1*) – driver, používající signály STEP a DIR, obsazuje dva výstupní piny. Ve výchozím stavu je signálu STEP přiřazen pin 2 a signálu DIR pin 3. V případě použití vstupu ENABLE, použijte po dokončení inicializace objektu funkci *setEnablePin()*.

Logické úrovně pinů můžete invertovat funkcí *setPinsInverted()*.

**AccelStepper::FULL2WIRE** (*= 2*) – driver s negací signálu pro dvoufázový motor v režimu full-step, obsazuje dva výstupní piny

**AccelStepper::FULL3WIRE** (*= 3*) – driver pro trojfázový motor (např. vřeteno HDD) v režimu full-step, obsazuje tři výstupní piny

**AccelStepper::FULL4WIRE** (*= 4*) – driver pro dvoufázový motor v režimu full-step, obsazuje čtyři výstupní piny.

**AccelStepper::HALF3WIRE** (*= 6*) – driver pro trojfázový motor (např. vřeteno HDD) v režimu half-step, obsazuje tři výstupní piny

**AccelStepper::HALF4WIRE** (*= 8*) – driver pro dvoufázový motor v režimu half-step, obsazuje čtyři výstupní piny

## Příklad:

**Konstruktor ve formátu:**

```
AccelStepper stepper1(AccelStepper::DRIVER, 2, 3);
```

vytvoří instanci třídy AccelStepper, pojmenovanou *stepper1* v konfiguraci pro driver, řízený signály **STEP** (pin 2) a **DIR** (pin 3).

### Konstruktor ve formátu:

```
AccelStepper stepper2(AccelStepper::FULL4WIRE, 2, 3, 4, 5);
```

vytvoří instanci třídy AccelStepper, pojmenovanou *stepper2* v konfiguraci pro přímé řízení dvoufázového krokového motoru s výkonovými spínači připojenými na piny 2 až 5.