Python
STC1003_0 - 4A Stepper Phidget 121
Stepper API

# Phidget22.Devices.Stepper extends Phidget

The Stepper class powers and controls the stepper motor connected to the Phidget controller, allowing you to change the position, velocity, acceleration, and current limit.

## *Methods:*

### getAcceleration()

*Default:* ***10 000.0***

#### getAcceleration()

The rate at which the controller can change the motor's `Velocity`.

- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.
- Changing the acceleration value while the stepper is in motion (especially at speeds higher than 1000 1/16th steps/s) can cause unpredictable results due to the inability of the processor tocalculate a new acceleration curve quickly enough. Generally you should wait until the motor is stationary until calling this function.

Returns:

The acceleration value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

acceleration = ch.getAcceleration()
print("Acceleration: " + str(acceleration))

ch.close()
```

### setAcceleration()

### setAcceleration(**acceleration**)

The rate at which the controller can change the motor's `Velocity`.

- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.
- Changing the acceleration value while the stepper is in motion (especially at speeds higher than 1000 1/16th steps/s) can cause unpredictable results due to the inability of the processor tocalculate a new acceleration curve quickly enough. Generally you should wait until the motor is stationary until calling this function.

Parameters:

**Acceleration** (type: `float`): The acceleration value

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setAcceleration(10000)

ch.close()
```

---

## getMinAcceleration()

*Constant:* ***2.0***

### getMinAcceleration()

The minimum value that `Acceleration` can be set to.

- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The acceleration value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

minAcceleration = ch.getMinAcceleration()
print("MinAcceleration: " + str(minAcceleration))

ch.close()
```

## getMaxAcceleration()

*Constant:* ***10 000 000.0***

### getMaxAcceleration()

The maximum value that `Acceleration` can be set to.

- Units for `Position` , `Velocity` , and `Acceleration` can be set by the user through the `RescaleFactor` .The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The acceleration value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

maxAcceleration = ch.getMaxAcceleration()
print("MaxAcceleration: " + str(maxAcceleration))

ch.close()
```

## getControlMode()

*Default:* ***CONTROL_MODE_STEP***

### getControlMode()

Use `StepperControlMode.CONTROL_MODE_STEP` mode when you want to set a `TargetPosition` for the Stepper motor. Use `StepperControlMode.CONTROL_MODE_RUN` mode when you simply want the Stepper motor to rotate continuously in a specific direction.

Changing the control mode while the motor is running will not have an effect on the motor's movements until a

new `TargetPosition` or `VelocityLimit` is set.

In `StepperControlMode.CONTROL_MODE_RUN` mode, setting a `VelocityLimit` will also set the target position of the controller to `MinPosition` or `MaxPosition`, corresponding to the direction of movement.

Returns:

The control mode value (type: StepperControlMode)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

controlMode = ch.getControlMode()
print("ControlMode: " + str(controlMode))

ch.close()
```

---

## setControlMode()

### setControlMode(**controlMode**)

Use `StepperControlMode.CONTROL_MODE_STEP` mode when you want to set a `TargetPosition` for the Stepper motor. Use `StepperControlMode.CONTROL_MODE_RUN` mode when you simply want the Stepper motor to rotate continuously in a specific direction.

Changing the control mode while the motor is running will not have an effect on the motor's movements until a new `TargetPosition` or `VelocityLimit` is set.

In `StepperControlMode.CONTROL_MODE_RUN` mode, setting a `VelocityLimit` will also set the target position of the controller to `MinPosition` or `MaxPosition`, corresponding to the direction of movement.

Parameters:

**ControlMode** (type: StepperControlMode): The control mode value

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setControlMode(StepperControlMode.CONTROL_MODE_STEP)

ch.close()
```

---

## getCurrentLimit()

*Default: **1.0** A*

### getCurrentLimit()

The current through the motor will be limited by the `CurrentLimit`.

- See your Stepper motor's data sheet for more information about what value the `CurrentLimit` should be.

Returns:

The current limit value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

currentLimit = ch.getCurrentLimit()
print("CurrentLimit: " + str(currentLimit))

ch.close()
```

---

## setCurrentLimit()

### setCurrentLimit(**currentLimit**)

The current through the motor will be limited by the `CurrentLimit`.

- See your Stepper motor's data sheet for more information about what value the `CurrentLimit` should be.

Parameters:

**`CurrentLimit`** (type: `float`): The current limit value

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setCurrentLimit(1)

ch.close()
```

---

## getMinCurrentLimit()

*Constant: **0.0** A*

### getMinCurrentLimit()

The minimum value that `CurrentLimit` and `HoldingCurrentLimit` can be set to.

- Reference your controller's User Guide for more information about how the `HoldingCurrentLimit` and `CurrentLimit` can be used in your application.

Returns:

The current limit (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

minCurrentLimit = ch.getMinCurrentLimit()
print("MinCurrentLimit: " + str(minCurrentLimit))

ch.close()
```

---

## getMaxCurrentLimit()

*Constant: **4.0** A*

### getMaxCurrentLimit()

The maximum value that `CurrentLimit` and `HoldingCurrentLimit` can be set to.

- Reference your controller's User Guide for more information about how the `HoldingCurrentLimit` and `CurrentLimit` can be used in your application.

Returns:

The current limit (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

maxCurrentLimit = ch.getMaxCurrentLimit()
print("MaxCurrentLimit: " + str(maxCurrentLimit))

ch.close()
```

---

## getDataInterval()

*Default: **250** ms*

### getDataInterval()

The `DataInterval` is the time that must elapse before the channel will fire another `PositionChange` / `VelocityChange` event.

- The data interval is bounded by `MinDataInterval` and `MaxDataInterval`.

Returns:

The data interval value (type: `int`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

dataInterval = ch.getDataInterval()
print("DataInterval: " + str(dataInterval))

ch.close()
```

---

## setDataInterval()

### setDataInterval(**dataInterval**)

The `DataInterval` is the time that must elapse before the channel will fire another `PositionChange` / `VelocityChange` event.

- The data interval is bounded by `MinDataInterval` and `MaxDataInterval`.

Parameters:

**`DataInterval`** (type: `int`): The data interval value

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setDataInterval(1000)

ch.close()
```

## getMinDataInterval()

*Constant: **100** ms*

### getMinDataInterval()

The minimum value that `DataInterval` can be set to.

Returns:

The data interval value (type: `int`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

minDataInterval = ch.getMinDataInterval()
print("MinDataInterval: " + str(minDataInterval))

ch.close()
```

## getMaxDataInterval()

*Constant: **60 000** ms*

### getMaxDataInterval()

The maximum value that `DataInterval` can be set to.

Returns:

The data interval value (type: `int`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

maxDataInterval = ch.getMaxDataInterval()
print("MaxDataInterval: " + str(maxDataInterval))

ch.close()
```

---

# getDataRate()

*Unit: hertz (Hz)*

## getDataRate()

The `DataRate` is the frequency of events from the device.

- The data rate is bounded by `MinDataRate` and `MaxDataRate`.
- Changing `DataRate` will change the channel's `DataInterval` to a corresponding value, rounded to the nearest integer number of milliseconds.
- The timing between events can also affected by the change trigger.

Returns:

The data rate for the channel (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

dataRate = ch.getDataRate()
print("DataRate: " + str(dataRate))

ch.close()
```

---

# setDataRate()

## setDataRate(**dataRate**)

The `DataRate` is the frequency of events from the device.

- The data rate is bounded by `MinDataRate` and `MaxDataRate`.
- Changing `DataRate` will change the channel's `DataInterval` to a corresponding value, rounded to the nearest integer number of milliseconds.

- The timing between events can also affected by the change trigger.

Parameters:

**DataRate** (type: `float`): The data rate for the channel

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setDataRate(1000)

ch.close()
```

---

## getMinDataRate()

*Unit: hertz (Hz)*

### getMinDataRate()

The minimum value that `DataRate` can be set to.

Returns:

The data rate value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

minDataRate = ch.getMinDataRate()
print("MinDataRate: " + str(minDataRate))

ch.close()
```

## getMaxDataRate()

*Unit: hertz (Hz)*

### getMaxDataRate()

The maximum value that `DataRate` can be set to.

Returns:

The data rate value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

maxDataRate = ch.getMaxDataRate()
print("MaxDataRate: " + str(maxDataRate))

ch.close()
```

## getEngaged()

*Default: **false***

### getEngaged()

When this property is true, the controller will supply power to the motor coils.

- The controller must be `Engaged` in order to move the Stepper motor, or have it hold position.

Returns:

The engaged state (type: `boolean`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

engaged = ch.getEngaged()
print("Engaged: " + str(engaged))

ch.close()
```

---

# setEngaged()

## setEngaged(**engaged**)

When this property is true, the controller will supply power to the motor coils.

- The controller must be `Engaged` in order to move the Stepper motor, or have it hold position.

Parameters:

**Engaged** (type: boolean): The engaged state

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setEngaged(True)

ch.close()
```

---

# enableFailsafe()

## enableFailsafe(**failsafeTime**)

Enables the **failsafe** feature for the channel, with a given **failsafe time**.

The **failsafe** feature is intended for use in applications where it is important for the channel to enter a known *safe state* if the program controlling it locks up or crashes. If you do not enable the failsafe feature, the channel will carry out whatever instructions it was last given until it is explicitly told to stop.

Enabling the failsafe feature starts a recurring **failsafe timer** for the channel. Once the failsafe timer is enabled,

it must be reset within the specified time or the channel will enter a **failsafe state**. The failsafe timer may be reset by sending any valid command to the device*. Resetting the failsafe timer will reload the timer with the specified *failsafe time*, starting when the message to reset the timer is received by the Phidget.

*(***get** *requests do not typically send commands and won't reset the failsafe timer)*

For example: if the failsafe is enabled with a **failsafe time** of 1000ms, you will have 1000ms to reset the failsafe timer. Every time the failsafe timer is reset, you will have 1000ms from that time to reset the failsafe again.

If the failsafe timer is not reset before it runs out, the channel will enter a **failsafe state**. For Stepper Motor channels, this will disengage the motor. Once the channel enters the **failsafe state**, it will reject any further input until the channel is reopened.

To prevent the channel from falsely entering the failsafe state, we recommend resetting the failsafe timer as frequently as is practical for your application. A good rule of thumb is to not let more than a third of the failsafe time pass before resetting the timer.

Once the failsafe timer has been set, it cannot be disabled by any means other than closing and reopening the channel.

Parameters:

**`failsafeTime`** (type: `int`): Failsafe timeout in milliseconds

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.enableFailsafe(20000)

ch.close()
```

---

# getMinFailsafeTime()

*Constant: **500** ms*

## getMinFailsafeTime()

The minimum value that `failsafeTime` can be set to when calling `enableFailsafe()`.

Returns:

The failsafe time (type: `int`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

minFailsafeTime = ch.getMinFailsafeTime()
print("MinFailsafeTime: " + str(minFailsafeTime))

ch.close()
```

---

# getMaxFailsafeTime()

*Constant: **30 000** ms*

## getMaxFailsafeTime()

The maximum value that `failsafeTime` can be set to when calling `enableFailsafe()` .

Returns:

The failsafe time (type: `int`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

maxFailsafeTime = ch.getMaxFailsafeTime()
print("MaxFailsafeTime: " + str(maxFailsafeTime))

ch.close()
```

---

# getHoldingCurrentLimit()

*Unit: amperes (A)*

## getHoldingCurrentLimit()

The `HoldingCurrentLimit` will activate when the `TargetPosition` has been reached. It will limit current through the motor.

- When the motor is not stopped, the current through the motor is limited by the `CurrentLimit` .
- If no `HoldingCurrentLimit` is specified, the `CurrentLimit` value will persist when the motor is stopped.
- Reference your controller's User Guide for more information about how the `HoldingCurrentLimit` and `CurrentLimit` can be used in your application.

Returns:

The current value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

holdingCurrentLimit = ch.getHoldingCurrentLimit()
print("HoldingCurrentLimit: " + str(holdingCurrentLimit))

ch.close()
```

---

## setHoldingCurrentLimit()

### setHoldingCurrentLimit(**holdingCurrentLimit**)

The `HoldingCurrentLimit` will activate when the `TargetPosition` has been reached. It will limit current through the motor.

- When the motor is not stopped, the current through the motor is limited by the `CurrentLimit`.
- If no `HoldingCurrentLimit` is specified, the `CurrentLimit` value will persist when the motor is stopped.
- Reference your controller's User Guide for more information about how the `HoldingCurrentLimit` and `CurrentLimit` can be used in your application.

Parameters:

**`HoldingCurrentLimit`** (type: `float`): The current value

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setHoldingCurrentLimit(1)

ch.close()
```

---

# getIsMoving()

## getIsMoving()

`IsMoving` returns true while the controller is sending commands to the motor. Note: there is no feedback to the controller, so it does not know whether the motor shaft is actually moving or not.

Returns:

The moving state (type: boolean)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

isMoving = ch.getIsMoving()
print("IsMoving: " + str(isMoving))

ch.close()
```

---

# getPosition()

## getPosition()

The most recent position value that the controller has reported.

- This value will always be between `MinPosition` and `MaxPosition`.
- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The position value (type: float)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

position = ch.getPosition()
print("Position: " + str(position))

ch.close()
```

---

## getMinPosition()

*Constant:* ***-1 000 000 000 000 000.0***

### getMinPosition()

The minimum value that `TargetPosition` can be set to.

- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`. The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The position value (type: float)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

minPosition = ch.getMinPosition()
print("MinPosition: " + str(minPosition))

ch.close()
```

---

## getMaxPosition()

*Constant:* ***1 000 000 000 000 000.0***

### getMaxPosition()

The maximum value that `TargetPosition` can be set to.

- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The position value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

maxPosition = ch.getMaxPosition()
print("MaxPosition: " + str(maxPosition))

ch.close()
```

---

## addPositionOffset()

### addPositionOffset(**positionOffset**)

Adds an offset (positive or negative) to the current position and target position.

- This is especially useful for zeroing position.

Parameters:

**positionOffset** (type: `float`): Amount to offset the position by

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.addPositionOffset(1000)

ch.close()
```

---

## getRescaleFactor()

*Default:* ***1.0***

### getRescaleFactor()

Applies a factor to the [user units] per step to all movement parameters to make the units in your application is more intuitive.

- For example, starting from position 0 and setting a new position with a rescale factor, the stepper will move `Position / RescaleFactor` steps.
- In this way, units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`. The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

```
RescaleFactor = (1/16) * (MotorStepAngle/Degrees Per UserUnit)
```

Returns:

The rescale factor value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

rescaleFactor = ch.getRescaleFactor()
print("RescaleFactor: " + str(rescaleFactor))

ch.close()
```

---

## setRescaleFactor()

### setRescaleFactor(**rescaleFactor**)

Applies a factor to the [user units] per step to all movement parameters to make the units in your application is more intuitive.

- For example, starting from position 0 and setting a new position with a rescale factor, the stepper will move `Position / RescaleFactor` steps.
- In this way, units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`. The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

```
RescaleFactor = (1/16) * (MotorStepAngle/Degrees Per UserUnit)
```

Parameters:

**`RescaleFactor`** (type: `float`): The rescale factor value

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setRescaleFactor(1)

ch.close()
```

---

# resetFailsafe()

## resetFailsafe()

Resets the failsafe timer, if one has been set. See `enableFailsafe()` for details.

This function will fail if no failsafe timer has been set for the channel.

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.resetFailsafe()

ch.close()
```

---

# getTargetPosition()

*Default:* ***0.0***

## getTargetPosition()

If the controller is configured and the `TargetPosition` is set, the Stepper motor will move towards the `TargetPosition` at the specified `Acceleration` and `Velocity`.

- `TargetPosition` is only used when the `ControlMode` is set to step mode.
- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The position value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

targetPosition = ch.getTargetPosition()
print("TargetPosition: " + str(targetPosition))

ch.close()
```

---

# setTargetPosition()

## setTargetPosition(**targetPosition**)

If the controller is configured and the `TargetPosition` is set, the Stepper motor will move towards the `TargetPosition` at the specified `Acceleration` and `Velocity`.

- `TargetPosition` is only used when the `ControlMode` is set to step mode.
- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the

`RescaleFactor` .The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Parameters:

**`TargetPosition`** (type: `float`): The position value

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setTargetPosition(10000)

ch.close()
```

---

# setTargetPosition_async()

## setTargetPosition_async(**targetPosition, asyncHandler**)

If the controller is configured and the `TargetPosition` is set, the Stepper motor will move towards the `TargetPosition` at the specified `Acceleration` and `Velocity` .
- `TargetPosition` is only used when the `ControlMode` is set to step mode.
- Units for `Position` , `Velocity` , and `Acceleration` can be set by the user through the `RescaleFactor` .The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Parameters:

**`TargetPosition`** (type: `float`): The position value
**`asyncHandler`** (type: `AsyncHandler(self, ErrorCode res, string details)`): Asynchronous completion callback

Returns:

No value is returned

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *
from Phidget22.ErrorCode import *
import time

ch = Stepper()
ch.openWaitForAttachment(1000)

def AsyncResult(ch, res, details):
    if res != ErrorCode.EPHIDGET_OK:
        print("Async failure: %i : %s" % (res, details))

ch.setTargetPosition_async(10000, AsyncResult)
# NOTE: Make sure to wait for async call to complete before closing the channel
time.sleep(1)

ch.close()
```

---

# getVelocity()

## getVelocity()

The most recent velocity value that the controller has reported.

- This value is bounded by `MinVelocityLimit` and `MaxVelocityLimit`.
- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The velocity value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

velocity = ch.getVelocity()
print("Velocity: " + str(velocity))

ch.close()
```

# getVelocityLimit()

*Default:* ***10 000.0***

## getVelocityLimit()

When moving, the Stepper motor velocity will be limited by this value.

- The `VelocityLimit` is bounded by `MinVelocityLimit` and `MaxVelocityLimit`.
- When in step mode, the `MinVelocityLimit` has a value of 0. This is because the sign (±) of the `TargetPosition` will indicate the direction.
- When in run mode, the `MinVelocityLimit` has a value of - `MaxVelocityLimit`. This is because there is no target position, so the direction is defined by the sign (±) of the `VelocityLimit`.
- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

Velocity limit (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

velocityLimit = ch.getVelocityLimit()
print("VelocityLimit: " + str(velocityLimit))

ch.close()
```

# setVelocityLimit()

## setVelocityLimit(**velocityLimit**)

When moving, the Stepper motor velocity will be limited by this value.

- The `VelocityLimit` is bounded by `MinVelocityLimit` and `MaxVelocityLimit`.
- When in step mode, the `MinVelocityLimit` has a value of 0. This is because the sign (±) of the `TargetPosition` will indicate the direction.
- When in run mode, the `MinVelocityLimit` has a value of - `MaxVelocityLimit`. This is because there is no target position, so the direction is defined by the sign (±) of the `VelocityLimit`.
- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Parameters:

**`VelocityLimit`** (type: `float`): Velocity limit

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

ch.setVelocityLimit(10000)

ch.close()
```

---

## getMinVelocityLimit()

*Constant:* ***0.0***

### getMinVelocityLimit()

The minimum value that `VelocityLimit` can be set to.

- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The velocity limit value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

minVelocityLimit = ch.getMinVelocityLimit()
print("MinVelocityLimit: " + str(minVelocityLimit))

ch.close()
```

---

## getMaxVelocityLimit()

*Constant:* ***115 000.0***

### getMaxVelocityLimit()

The maximum value that `VelocityLimit` can be set to.

- Units for `Position`, `Velocity`, and `Acceleration` can be set by the user through the `RescaleFactor`.The `RescaleFactor` allows you to use more intuitive units such as rotations, or degrees.
- The default units for this motor controller are **1/16steps per count**.

Returns:

The velocity value (type: `float`)

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *

ch = Stepper()
ch.openWaitForAttachment(1000)

maxVelocityLimit = ch.getMaxVelocityLimit()
print("MaxVelocityLimit: " + str(maxVelocityLimit))

ch.close()
```

## *Events:*

### setOnPositionChangeHandler()

#### setOnPositionChangeHandler(**handler**)

Assigns a handler that will be called when the PositionChange event occurs

Parameters:

**`handler`** (type: `PositionChangeHandler(self, position)`): The callback function

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *
import time

def onPositionChange(self, position):
    print("Position: " + str(position))

ch = Stepper()

# Register for event before calling open
ch.setOnPositionChangeHandler(onPositionChange)

ch.open()

while True:
    # Do work, wait for events, etc.
    time.sleep(1)
```

---

## setOnStoppedHandler()

### setOnStoppedHandler(**handler**)

Assigns a handler that will be called when the Stopped event occurs

Parameters:

**handler** (type: StoppedHandler(self)): The callback function

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *
import time

def onStopped(self):
    print("Stopped")

ch = Stepper()

# Register for event before calling open
ch.setOnStoppedHandler(onStopped)

ch.open()

while True:
    # Do work, wait for events, etc.
    time.sleep(1)
```

---

## setOnVelocityChangeHandler()

### setOnVelocityChangeHandler(**handler**)

Assigns a handler that will be called when the VelocityChange event occurs

Parameters:

**handler** (type: VelocityChangeHandler(self, velocity)): The callback function

Returns:

No value is returned

Throws:

PhidgetException

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *
import time

def onVelocityChange(self, velocity):
    print("Velocity: " + str(velocity))

ch = Stepper()

# Register for event before calling open
ch.setOnVelocityChangeHandler(onVelocityChange)

ch.open()

while True:
    # Do work, wait for events, etc.
    time.sleep(1)
```

---

## PositionChange

### void onPositionChange(**self, position**)

Occurs when the controller updates the stepper motor position.

- This event will still fire even if the motor is blocked from physically moving or misses steps.

**self** (type: Stepper): object which sent the event
**position** (type: float): The current stepper position

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *
import time

def onPositionChange(self, position):
    print("Position: " + str(position))

ch = Stepper()

# Register for event before calling open
ch.setOnPositionChangeHandler(onPositionChange)

ch.open()

while True:
    # Do work, wait for events, etc.
    time.sleep(1)
```

## Stopped

### void onStopped(**self**)

Occurs when the motor controller stops.

- The motor may still be physically moving if the inertia is great enough to make it misstep.

**self** (type: Stepper): object which sent the event

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *
import time

def onStopped(self):
    print("Stopped")

ch = Stepper()

# Register for event before calling open
ch.setOnStoppedHandler(onStopped)

ch.open()

while True:
    # Do work, wait for events, etc.
    time.sleep(1)
```

## VelocityChange

### void onVelocityChange(**self, velocity**)

Occurs when the stepper motor velocity changes.

**self** (type: Stepper): object which sent the event
**velocity** (type: float): Velocity of the stepper. Sign indicates direction.

Example:

```
from Phidget22.Phidget import *
from Phidget22.Devices.Stepper import *
import time

def onVelocityChange(self, velocity):
    print("Velocity: " + str(velocity))

ch = Stepper()

# Register for event before calling open
ch.setOnVelocityChangeHandler(onVelocityChange)

ch.open()

while True:
    # Do work, wait for events, etc.
    time.sleep(1)
```

## *Error Events:*

### BadPower

**Error Event code:** EEPHIDGET_BADPOWER **(Value: 0x1008) -** *Bad Power*

The controller will report a `BadPower` event when the power supply voltage is not within the valid range.

- This could be due to the stepper motor(s) drawing too much current.

### Failsafe

**Error Event code:** EEPHIDGET_FAILSAFE **(Value: 0x100c) -** *Failsafe*

The channel has entered its failsafe state. See `enableFailsafe()` for details.

- This occurs when the your program has enabled the **failsafe** for the channel, and failed to call `resetFailsafe()` in time.
- After you receive this message, the channel will reject any further communications until it has been reopened.

## *Enumerations:*

Phidget22.StepperControlMode *Method of motor control*

| Name | Value | Description |
| --- | --- | --- |
| CONTROL_MODE_STEP | 0x0 | *Step*<br>Control the motor by setting a target position. |
| CONTROL_MODE_RUN | 0x1 | *Run*<br>Control the motor by selecting a target velocity (sign indicates direction). The motor will rotate continuously in the chosen direction. |