Application Programming Interface: Python v4.6

Contents

Application Programming Interface: Python v4.6

Machine Motion V2

Overview

This is the developers reference manual for version 4.6 of MachineMotion's Python API. This API is compatible with Python version 2.7, 3.6 and later.
If this is your first time using the MachineMotion controller, please follow the Python programming manual here.
Version 4.6 of the API is compatible with MachineMotion controller software version 2.1 and later. However, some functions are only supported on newer software versions, as described below:

Function	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9
loadPath					✓	✓	✓	✓
startPath					✓	✓	✓	✓

Function	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9
stopPath					✓	✓	✓	✓
configActuator					✓	✓	✓	✓
setAxisMaxSpeed			✓	✓	✓	✓	✓	✓
setAxisMaxAcceleration			✓	✓	✓	✓	✓	✓
getAxisMaxSpeed			✓	✓	✓	✓	✓	✓
getAxisMaxAcceleration			✓	✓	✓	✓	✓	✓
getActualSpeeds			✓	✓	✓	✓	✓	✓
stopAxis			✓	✓	✓	✓	✓	✓
releaseEstop	✓	✓	✓	✓	✓	✓	✓	
resetSystem	✓	✓	✓	✓	✓	✓	✓	
triggerEstop	✓	✓	✓	✓	✓	✓	✓	

If your controller software is 1.2.11 or earlier, please refer to Python API v2.2.
To get your controller software version, refer to the user manual here.

Download

The Python API comes pre-installed on your MachineMotion controller. To update the API version or download it to an external computer, follow this link to version 4.6 of the Python API on Vention's Github.

MachineMotion v2 One-Drive

For use with MachineMotion 2 One-Drive (CE-CL-010-0001) please refer to the provided example which can be found here.

API Reference

Variables

```
DEFAULT_IP = DEFAULT_IP_ADDRESS.usb_windows
HARDWARE_MIN_HOMING_FEEDRATE = 500
HARDWARE_MAX_HOMING_FEEDRATE = 8000
MIN_MOTOR_CURRENT = 1.5
MAX_MOTOR_CURRENT = 10.0
ELECTRIC_CYLINDER_MAX_MOTOR_CURRENT = 1.68
MAX_IO_ADDRESS = 8
MIN_HOMING_SPEED = 1
DEFAULT_TIMEOUT = 65
```

class MachineMotion(object)

Variables

```
EXEC_ENGINE_PORT = 3100
```

__init__(self, machineIp, gCodeCallback, machineMotionHwVersion)

- **desc** Constructor of MachineMotion class
  - **params**
    - **machineIp**
      - **desc** IP address of the Machine Motion
      - **type** string of the DEFAULT_IP_ADDRESS class, or other valid IP address
      - **default** DEFAULT_IP_ADDRESS.usb_windows
    - **gCodeCallback**
      - **desc** Allows to define a custom behaviour for the MachineMotion objectwhen a response is received from the motion controller.
      - **type** function
      - **default** None
    - **machineMotionHwVersion**
      - **desc** The hardware version of the MachineMotion being used
      - **type** string of the MACHINEMOTION_HW_VERSIONS class
      - **default** MACHINEMOTION_HW_VERSIONS.MMv1
  - **compatibility** MachineMotion v1 and MachineMotion v2.

  Example: moveRelative.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases relative moves with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Begin Relative Move
distance = 100  # in mm
mm.moveRelative(axis, distance)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " moved " + str(distance) + "mm")

# Pass a negative distance value to move in the opposite direction
distance = -100  # in mm
mm.moveRelative(axis, distance)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " moved " + str(distance) + "mm")

print("--> Example Complete")
```

Example: oneDriveControl.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to control a motor with a One Drive MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

### MachineMotion configuration ###
mm = MachineMotionV2OneDrive()

axis = AXIS_NUMBER.DRIVE1
motorCurrent = 10  # current (A)

### Home axis 1 specifically
print("--> Homing axis " + str(axis))
mm.moveToHome(axis)
mm.waitForMotionCompletion()

### Relative move axis 1
print("--> Moving axis " + str(axis) + " by 100mm.")
mm.moveRelative(axis, 100)
mm.waitForMotionCompletion()

### Absolute move axis 1
position = 50  # in mm
mm.moveToPosition(axis, position)
print("--> Axis " + str(axis) + " is moving towards position " + str(position) + "mm")
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " is at position " + str(position) + "mm")

### Getting position for axis 1
print("--> Read the position of axis " + str(axis))
actualPosition = mm.getActualPositions(axis)
print("--> Actual position is: " + str(actualPosition))

### Controlling any other axis will yield an error:
try:
    print("--> Controlling an axis that doesn't exist..")
    mm.moveToPosition(2, position)
except Exception as e:
    print(e)

print("Example Complete!")
```

moveContinuous(self, axis, speed, accel, direction)

- **desc** Starts an axis using speed mode.
  - **params**
    - **axis**
      - **desc** Axis to move
      - **type** Number of AXIS_NUMBER class

- **speed**
  - **desc** Optional. Speed to move the axis at in mm/s. If no speed is specified, the axis will move at the configured axis max speed in the positive axis direction.
  - **type** Number, positive or negative
- **accel**
  - **desc** Optional. Acceleration used to reach the desired speed, in mm/s^2. If no acceleration is specified, the axis will move at the configured axis max acceleration
  - **type** Positive number
- **direction**
  - **desc** Optional. Overrides sign of speed. Determines the direction of the continuous move. If no direction is given, the actuator will move according to the direction of speed.
  - **type** String of the DIRECTION class.

- **compatibility** MachineMotion v1 and MachineMotion v2. On software versions older than v2.4.0, speed and acceleration arguments are mandatory.

Example: moveContinuous.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases continuous moves with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Heavy Duty Roller Conveyor
#   Motor Size: Large Servo


# Create MachineMotion instance
mm = MachineMotionV2()

conveyor_axis = AXIS_NUMBER.DRIVE1

# Note: on MachineMotion software version 2.4.0 and newer, continuous moves are limited by the axis max speed and acceleration.
# Please see the section below for more details.

### CONTINUOUS MOVES ###
# Start the continuous move
print("Start Conveyor Move...")
print("Continuous move: speed 100mm/s & acceleration 100mm/s^2")
mm.moveContinuous(conveyor_axis, 100, 100)
time.sleep(5)

# Change speed while moving
print("Continuous move: speed 500mm/s & acceleration 250mm/s^2")
mm.moveContinuous(conveyor_axis, 500, 250)
time.sleep(5)

# Reverse direction of conveyor by changing the sign of the speed
print("Reverse continuous move: speed -1000mm/s & acceleration 500mm/s^2")
mm.moveContinuous(conveyor_axis, -1000, 500)
time.sleep(5)

# Or pass in the optional direction argument
print("Reverse continuous move: speed -500mm/s & acceleration 500mm/s^2")
mm.moveContinuous(conveyor_axis, 500, 500, direction=DIRECTION.NEGATIVE)
```

```
time.sleep(5)

# Stop the continuous move
print("Stop Conveyor Move: acceleration 500mm/s^2")
mm.stopMoveContinuous(conveyor_axis, 500)
time.sleep(3)

# # For MachineMotion Software version >= 2.4.0 only:

# # Setting the axis max speed and acceleration will limit continuous moves:
# mm.setAxisMaxSpeed(conveyor_axis, 1000)
# mm.setAxisMaxAcceleration(conveyor_axis, 1000)

# # Start the continuous move at max speed and acceleration
# mm.moveContinuous(conveyor_axis)
# time.sleep(5)

# # Slow down conveyor
# mm.setAxisMaxSpeed(conveyor_axis, 500)
# time.sleep(5)

# # Reverse direction of conveyor
# mm.moveContinuous(conveyor_axis, direction=DIRECTION.NEGATIVE)
# time.sleep(5)

# # Stop the continuous move
# mm.stopMoveContinuous(conveyor_axis)
# time.sleep(5)

print("--> Example completed")
```

stopMoveContinuous(self, axis, accel)

- **desc** Stops an axis using speed mode.
  - **params**
    - **axis**
      - **desc** Axis to move
      - **type** Number of the AXIS_NUMBER class
    - **accel**
      - **desc** Optional. Acceleration used to reach a null speed, in mm/s2. If no accel is specified, the axis will decelerate at the configured max axis acceleration.
      - **type** Positive number
  - **compatibility** MachineMotion v1 and MachineMotion v2. On software versions older than v2.4.0, accel argument is mandatory.

Example: moveContinuous.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases continuous moves with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
```

```
#   Drive Number(s): 1
#   Axis Type: Heavy Duty Roller Conveyor
#   Motor Size: Large Servo


# Create MachineMotion instance
mm = MachineMotionV2()

conveyor_axis = AXIS_NUMBER.DRIVE1

# Note: on MachineMotion software version 2.4.0 and newer, continuous moves are limited by the axis max speed and acceleration.
# Please see the section below for more details.

### CONTINUOUS MOVES ###
# Start the continuous move
print("Start Conveyor Move...")
print("Continuous move: speed 100mm/s & acceleration 100mm/s^2")
mm.moveContinuous(conveyor_axis, 100, 100)
time.sleep(5)

# Change speed while moving
print("Continuous move: speed 500mm/s & acceleration 250mm/s^2")
mm.moveContinuous(conveyor_axis, 500, 250)
time.sleep(5)

# Reverse direction of conveyor by changing the sign of the speed
print("Reverse continuous move: speed -1000mm/s & acceleration 500mm/s^2")
mm.moveContinuous(conveyor_axis, -1000, 500)
time.sleep(5)

# Or pass in the optional direction argument
print("Reverse continuous move: speed -500mm/s & acceleration 500mm/s^2")
mm.moveContinuous(conveyor_axis, 500, 500, direction=DIRECTION.NEGATIVE)
time.sleep(5)

# Stop the continuous move
print("Stop Conveyor Move: acceleration 500mm/s^2")
mm.stopMoveContinuous(conveyor_axis, 500)
time.sleep(3)

# # For MachineMotion Software version >= 2.4.0 only:

# # Setting the axis max speed and acceleration will limit continuous moves:
# mm.setAxisMaxSpeed(conveyor_axis, 1000)
# mm.setAxisMaxAcceleration(conveyor_axis, 1000)

# # Start the continuous move at max speed and acceleration
# mm.moveContinuous(conveyor_axis)
# time.sleep(5)

# # Slow down conveyor
# mm.setAxisMaxSpeed(conveyor_axis, 500)
# time.sleep(5)

# # Reverse direction of conveyor
# mm.moveContinuous(conveyor_axis, direction=DIRECTION.NEGATIVE)
# time.sleep(5)

# # Stop the continuous move
# mm.stopMoveContinuous(conveyor_axis)
# time.sleep(5)

print("--> Example completed")
```

getDesiredPositions(self, axis)

- **desc** Returns the desired position of the axes.
  - **params**
    - **axis (optional)**
      - **desc** The axis to get the desired position of.
      - **type** Number
  - **returnValue** The position of the axis if that parameter was specified, or a dictionary containing the desired position of every axis.
  - **returnValueType** Number or Dictionary of numbers
  - **note** This function returns the 'open loop' position of each axis.
  - **compatibility** Recommended for MachineMotion v1.

Example: getPositions.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to read actuator positions with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 2, 3, 4
#   Any valid configuration

mm = MachineMotionV2()

# Configure actuator
axis = AXIS_NUMBER.DRIVE1

# Home Axis Before Moving
print("--> Axis " + str(axis) + " moving home")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " homed")


###### READ POSITIONS ######

# Read the position of one axis
print("--> Read the position of one axis")
actualPosition_axis = mm.getActualPositions(axis)
print(
    "Actual position of axis "
    + str(axis)
    + " is : "
    + str(actualPosition_axis)
    + " mm."
)

# Read the position of several axes
print("--> Read the position of several axes")
actualPositions = mm.getActualPositions()
print("Actual position of axis 1 is : " + str(actualPositions[1]) + " mm.")
```

```
print("Actual position of axis 2 is : " + str(actualPositions[2]) + " mm.")
print("Actual position of axis 3 is : " + str(actualPositions[3]) + " mm.")
print("Actual position of axis 4 is : " + str(actualPositions[4]) + " mm.")

###### MOVE AND READ POSITIONS ######

# Define Motion Parameters
distance = 100

# Move 100mm and check position again
mm.moveRelative(axis, distance)
print("--> Move ongoing")
while not mm.isMotionCompleted():
    actualPosition_axis = mm.getActualPositions(axis)
    print(
        "Actual position of axis "
        + str(axis)
        + " is : "
        + str(actualPosition_axis)
        + " mm."
    )

mm.waitForMotionCompletion()
print("--> Move completed")
actualPosition_axis = mm.getActualPositions(axis)
print(
    "Actual position of axis "
    + str(axis)
    + " is : "
    + str(actualPosition_axis)
    + " mm."
)

print("--> End of example")
```

getActualPositions(self, axis)

- **desc** Returns the current position of the axes.
  - **params**
    - **axis (optional)**
      - **desc** The axis to get the current position of.
      - **type** Number
  - **returnValue** The position of the axis if that parameter was specified, or a dictionary containing the current position of every axis, in mm.
  - **returnValueType** Number or Dictionary of numbers
  - **compatibility** MachineMotion v1 and MachineMotion v2.

Example: getPositions.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to read actuator positions with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
```

```
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 2, 3, 4
#   Any valid configuration

mm = MachineMotionV2()

# Configure actuator
axis = AXIS_NUMBER.DRIVE1

# Home Axis Before Moving
print("--> Axis " + str(axis) + " moving home")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " homed")


###### READ POSITIONS ######

# Read the position of one axis
print("--> Read the position of one axis")
actualPosition_axis = mm.getActualPositions(axis)
print(
    "Actual position of axis "
    + str(axis)
    + " is : "
    + str(actualPosition_axis)
    + " mm."
)

# Read the position of several axes
print("--> Read the position of several axes")
actualPositions = mm.getActualPositions()
print("Actual position of axis 1 is : " + str(actualPositions[1]) + " mm.")
print("Actual position of axis 2 is : " + str(actualPositions[2]) + " mm.")
print("Actual position of axis 3 is : " + str(actualPositions[3]) + " mm.")
print("Actual position of axis 4 is : " + str(actualPositions[4]) + " mm.")

###### MOVE AND READ POSITIONS ######

# Define Motion Parameters
distance = 100

# Move 100mm and check position again
mm.moveRelative(axis, distance)
print("--> Move ongoing")
while not mm.isMotionCompleted():
    actualPosition_axis = mm.getActualPositions(axis)
    print(
        "Actual position of axis "
        + str(axis)
        + " is : "
        + str(actualPosition_axis)
        + " mm."
    )

mm.waitForMotionCompletion()
print("--> Move completed")
actualPosition_axis = mm.getActualPositions(axis)
print(
    "Actual position of axis "
    + str(axis)
    + " is : "
    + str(actualPosition_axis)
```

```
    + " mm."
)

print("--> End of example")
```

getEndStopState(self)

- **desc** Returns the current state of all home and end sensors.
  - **returnValue** The states of all end stop sensors {x_min, x_max, y_min, y_max, z_min, z_max} “TRIGGERED” or “open”
  - **returnValueType** Dictionary
  - **compatibility** MachineMotion v1 and MachineMotion v2.

  Example: getEndStopState.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to read enstop sensor states with MachineMotion v2. ###

mm = MachineMotionV2()

# Get End Stop State
endstopStates = mm.getEndStopState()
# If the direction of the axis is normal,
#   then the home sensor is the "_min" sensor, and the end sensor is the "_max" sensor.
# If the direction of the axis reversed,
#   then the home sensor is the "_max" sensor, and the end sensor is the "_min" sensor.
axis1_home_sensor_status = endstopStates["x_min"]
axis1_endstop_sensor_status = endstopStates["x_max"]
axis2_home_sensor_status = endstopStates["y_min"]
axis2_endstop_sensor_status = endstopStates["y_max"]
axis3_home_sensor_status = endstopStates["z_min"]
axis3_endstop_sensor_status = endstopStates["z_max"]
axis4_home_sensor_status = endstopStates["w_min"]
axis4_endstop_sensor_status = endstopStates["w_max"]

print("Axis 1 : " + "Home sensor is : " + str(axis1_home_sensor_status))
print("Axis 1 : " + "End sensor is : " + str(axis1_endstop_sensor_status))
print("Axis 2 : " + "Home sensor is : " + str(axis2_home_sensor_status))
print("Axis 2 : " + "End sensor is : " + str(axis2_endstop_sensor_status))
print("Axis 3 : " + "Home sensor is : " + str(axis3_home_sensor_status))
print("Axis 3 : " + "End sensor is : " + str(axis3_endstop_sensor_status))
print("Axis 4 : " + "Home sensor is : " + str(axis4_home_sensor_status))
print("Axis 4 : " + "End sensor is : " + str(axis4_endstop_sensor_status))
```

stopAllMotion(self)

- **desc** Immediately stops all motion of all axes.
  - **note** This function is a hard stop. It is not a controlled stop and consequently does not decelerate smoothly to a stop. Additionally, this function is not intended to serve as an emergency stop since this stop mechanism does not have safety ratings.

- **compatibility** MachineMotion v1 and MachineMotion v2.

Example: stopAllMotion.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to stop motion on MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Begin Relative Move
distance = 1000
mm.moveRelative(axis, distance)
print("Axis " + str(axis) + " is moving " + str(distance) + "mm")
# This move should take 5 seconds to complete (distance/speed). Instead, we wait 2 seconds and then stop the machine.
time.sleep(2)
mm.stopAllMotion()
print("Axis " + str(axis) + " stopped.")
```

stopAxis(self)

- **desc** Immediately stops motion on a set of axes.
  - **params**
    - ***axes**
      - **desc** Axis or axes to stop.
      - **type** Number or numbers separated by a comma, of the AXIS_NUMBER class
  - **compatibility** MachineMotion v2.

  Example: moveIndependentAxis.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### This Python example configures and moves an independent axis.
### The individual axis speed and acceleration are set,
### then motion on the axis is selectively started, stopped and monitored

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large Servo

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Move, stop and monitor the axis
axisSpeed = 50  # mm/s
axisAcceleration = 50  # mm/s^2
distance = 250  # mm

print(
    "Configuring axis: ",
    axis,
    " speed: ",
    axisSpeed,
    "mm/s",
    " acceleration: ",
    axisAcceleration,
    "mm/s^2",
)
mm.setAxisMaxSpeed(axis, axisSpeed)
mm.setAxisMaxAcceleration(axis, axisAcceleration)

print("Relative move: ", distance, "mm")
mm.moveRelative(axis, distance)

axisMotionCompleted = mm.isMotionCompleted(axis)
if axisMotionCompleted:
    print("Axis ", axis, " is NOT currently moving.")
else:
    print("Axis ", axis, " is currently moving.")

time.sleep(3)

print("Stopping just axis: ", axis)
mm.stopAxis(axis)

time.sleep(3)

print("Moving and waiting for axis: ", axis)
mm.moveToPosition(axis, distance)
mm.waitForMotionCompletion(axis)

print("Done!")
```

moveToHomeAll(self)

- **desc** Initiates the homing sequence of all axes. All axes will move home simultaneously on MachineMotion v2, and sequentially on MachineMotion v1.
  - **compatibility** MachineMotion v1 and MachineMotion v2.

  Example: moveToHomeAll.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to home actuators with MachineMotion v2. ###

### MachineMotion configuration ###

mm = MachineMotionV2()

### Home all axes simultaneously
print("All axes moving home simultaneously")
mm.moveToHomeAll()
mm.waitForMotionCompletion()
print("All axes homed")
```

moveToHome(self, axis)

- **desc** Initiates the homing sequence for the specified axis.
  - **params**
    - **axis**
      - **desc** The axis to be homed.
      - **type** Number
  - **note** If configAxisDirection is set to “normal” on axis 1, axis 1 will home itself towards sensor 1A. If configAxisDirection is set to “reverse” on axis 1, axis 1 will home itself towards sensor 1B.
  - **compatibility** MachineMotion v1 and MachineMotion v2.

  Example: moveToHome.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to home actuators with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Home the actuator
print("Axis " + str(axis) + " is going home")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " is at home")
```

setAxisMaxSpeed(self, axis, speed)

- **desc** Set maximum speed for a given axis.
  - **params**
    - **axis**
      - **desc** The target to set axis speed.
      - **type** Number of the AXIS_NUMBER class.
    - **speed**
      - **desc** The axis speed in mm/s.
      - **type** Number
  - **compatibility** MachineMotion v2.

  Example: speedAndAcceleration.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### speedAndAcceleration.py
# This example shows how to change the configured max speed and max acceleration for an axis,
# as well as how to retrieve the configured max speed, max acceleration and actual speed of a specific axis.

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

maxSpeedAxis = 50  # mm/s
maxAccelerationAxis = 150  # mm/s^2

# Setting speeds and accelerations

print("Setting max speed for axis ", axis, ": ", maxSpeedAxis, "mm/s")
mm.setAxisMaxSpeed(axis, maxSpeedAxis)
print("Setting max acceleration for axis ", axis, ": ", maxAccelerationAxis, "mm/s^2")
mm.setAxisMaxAcceleration(axis, maxAccelerationAxis)

# Getting speeds and accelerations

print("Getting speeds and accelerations...")

maxSpeed = mm.getAxisMaxSpeed(axis)
maxAccel = mm.getAxisMaxAcceleration(axis)
print(
    "For axis ",
    axis,
    " retrieved max speed: ",
    maxSpeed,
    "mm/s, max acceleration: ",
    maxAccel,
    "mm/s^2",
)

actualSpeed = mm.getActualSpeeds(axis)
print(
    "For axis ", axis, " retrieved actual speed: ", actualSpeed, "mm/s"
)  # The value should be 0 as the axis is not moving.

print("Getting actual speeds for all axes, in mm/s:")
actualSpeeds = mm.getActualSpeeds()
print(actualSpeeds)  # The values should be 0 as the axes are not moving.

print("Done!")
```

Example: moveIndependentAxis.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### This Python example configures and moves an independent axis.
### The individual axis speed and acceleration are set,
### then motion on the axis is selectively started, stopped and monitored

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large Servo

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Move, stop and monitor the axis
axisSpeed = 50  # mm/s
axisAcceleration = 50  # mm/s^2
distance = 250  # mm

print(
    "Configuring axis: ",
    axis,
    " speed: ",
    axisSpeed,
    "mm/s",
    " acceleration: ",
    axisAcceleration,
    "mm/s^2",
)
mm.setAxisMaxSpeed(axis, axisSpeed)
mm.setAxisMaxAcceleration(axis, axisAcceleration)

print("Relative move: ", distance, "mm")
mm.moveRelative(axis, distance)

axisMotionCompleted = mm.isMotionCompleted(axis)
if axisMotionCompleted:
    print("Axis ", axis, " is NOT currently moving.")
else:
    print("Axis ", axis, " is currently moving.")

time.sleep(3)

print("Stopping just axis: ", axis)
mm.stopAxis(axis)

time.sleep(3)

print("Moving and waiting for axis: ", axis)
mm.moveToPosition(axis, distance)
mm.waitForMotionCompletion(axis)

print("Done!")
```

getAxisMaxSpeed(self, axis)

- **desc** Get maximum speed for a given axis.
  - **params**
    - **axis**
      - **desc** The target axis to read maximum speed from.
      - **type** Number of the AXIS_NUMBER class.
  - **compatibility** MachineMotion v2.
  - **returnValue** Axis max speed in mm/s
  - **returnValueType** Number

  Example: speedAndAcceleration.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### speedAndAcceleration.py
# This example shows how to change the configured max speed and max acceleration for an axis,
# as well as how to retrieve the configured max speed, max acceleration and actual speed of a specific axis.

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

maxSpeedAxis = 50  # mm/s
maxAccelerationAxis = 150  # mm/s^2

# Setting speeds and accelerations

print("Setting max speed for axis ", axis, ": ", maxSpeedAxis, "mm/s")
mm.setAxisMaxSpeed(axis, maxSpeedAxis)
print("Setting max acceleration for axis ", axis, ": ", maxAccelerationAxis, "mm/s^2")
mm.setAxisMaxAcceleration(axis, maxAccelerationAxis)

# Getting speeds and accelerations

print("Getting speeds and accelerations...")

maxSpeed = mm.getAxisMaxSpeed(axis)
maxAccel = mm.getAxisMaxAcceleration(axis)
print(
    "For axis ",
    axis,
    " retrieved max speed: ",
    maxSpeed,
    "mm/s, max acceleration: ",
    maxAccel,
    "mm/s^2",
)

actualSpeed = mm.getActualSpeeds(axis)
print(
    "For axis ", axis, " retrieved actual speed: ", actualSpeed, "mm/s"
)  # The value should be 0 as the axis is not moving.

print("Getting actual speeds for all axes, in mm/s:")
actualSpeeds = mm.getActualSpeeds()
print(actualSpeeds)  # The values should be 0 as the axes are not moving.

print("Done!")
```

getActualSpeeds(self, axis)

- **desc** Get the current measured speed of all or one axis, in mm/s
  - **params**
    - **axis**

- **desc** The axis to get the current speed of. If None is passed, all axes will be checked.
- **type** Number of the AXIS_NUMBER class, or None
- **default** None

- **compatibility** MachineMotion v2, software version 2.2.0 and newer.
- **returnValue** The current speed of the axis if that parameter was specified, or a dictionary containing the current speed of every axis, in mm/s.
- **returnValueType** Number or Dictionary of numbers

Example: speedAndAcceleration.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### speedAndAcceleration.py
# This example shows how to change the configured max speed and max acceleration for an axis,
# as well as how to retrieve the configured max speed, max acceleration and actual speed of a specific axis.

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

maxSpeedAxis = 50  # mm/s
maxAccelerationAxis = 150  # mm/s^2

# Setting speeds and accelerations

print("Setting max speed for axis ", axis, ": ", maxSpeedAxis, "mm/s")
mm.setAxisMaxSpeed(axis, maxSpeedAxis)
print("Setting max acceleration for axis ", axis, ": ", maxAccelerationAxis, "mm/s^2")
mm.setAxisMaxAcceleration(axis, maxAccelerationAxis)

# Getting speeds and accelerations

print("Getting speeds and accelerations...")

maxSpeed = mm.getAxisMaxSpeed(axis)
maxAccel = mm.getAxisMaxAcceleration(axis)
print(
    "For axis ",
    axis,
    " retrieved max speed: ",
    maxSpeed,
    "mm/s, max acceleration: ",
    maxAccel,
    "mm/s^2",
)

actualSpeed = mm.getActualSpeeds(axis)
print(
    "For axis ", axis, " retrieved actual speed: ", actualSpeed, "mm/s"
)  # The value should be 0 as the axis is not moving.

print("Getting actual speeds for all axes, in mm/s:")
actualSpeeds = mm.getActualSpeeds()
print(actualSpeeds)  # The values should be 0 as the axes are not moving.

print("Done!")
```

setAxisMaxAcceleration(self, axis, acceleration)

- **desc** Set maximum acceleration for a given axis.
  - **params**
    - **axis**
      - **desc** The target to set axis acceleration.
      - **type** Number of the AXIS_NUMBER class.
    - **acceleration**
      - **desc** The axis acceleration in mm/s^2
      - **type** Number
  - **compatibility** MachineMotion v2.

  Example: speedAndAcceleration.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### speedAndAcceleration.py
# This example shows how to change the configured max speed and max acceleration for an axis,
# as well as how to retrieve the configured max speed, max acceleration and actual speed of a specific axis.

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

maxSpeedAxis = 50  # mm/s
maxAccelerationAxis = 150  # mm/s^2

# Setting speeds and accelerations

print("Setting max speed for axis ", axis, ": ", maxSpeedAxis, "mm/s")
mm.setAxisMaxSpeed(axis, maxSpeedAxis)
print("Setting max acceleration for axis ", axis, ": ", maxAccelerationAxis, "mm/s^2")
mm.setAxisMaxAcceleration(axis, maxAccelerationAxis)

# Getting speeds and accelerations

print("Getting speeds and accelerations...")

maxSpeed = mm.getAxisMaxSpeed(axis)
maxAccel = mm.getAxisMaxAcceleration(axis)
print(
    "For axis ",
    axis,
    " retrieved max speed: ",
    maxSpeed,
    "mm/s, max acceleration: ",
    maxAccel,
    "mm/s^2",
)

actualSpeed = mm.getActualSpeeds(axis)
print(
    "For axis ", axis, " retrieved actual speed: ", actualSpeed, "mm/s"
)  # The value should be 0 as the axis is not moving.

print("Getting actual speeds for all axes, in mm/s:")
actualSpeeds = mm.getActualSpeeds()
print(actualSpeeds)  # The values should be 0 as the axes are not moving.

print("Done!")
```

Example: moveIndependentAxis.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### This Python example configures and moves an independent axis.
### The individual axis speed and acceleration are set,
### then motion on the axis is selectively started, stopped and monitored

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large Servo

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Move, stop and monitor the axis
axisSpeed = 50  # mm/s
axisAcceleration = 50  # mm/s^2
distance = 250  # mm

print(
    "Configuring axis: ",
    axis,
    " speed: ",
    axisSpeed,
    "mm/s",
    " acceleration: ",
    axisAcceleration,
    "mm/s^2",
)
mm.setAxisMaxSpeed(axis, axisSpeed)
mm.setAxisMaxAcceleration(axis, axisAcceleration)

print("Relative move: ", distance, "mm")
mm.moveRelative(axis, distance)

axisMotionCompleted = mm.isMotionCompleted(axis)
if axisMotionCompleted:
    print("Axis ", axis, " is NOT currently moving.")
else:
    print("Axis ", axis, " is currently moving.")

time.sleep(3)

print("Stopping just axis: ", axis)
mm.stopAxis(axis)

time.sleep(3)

print("Moving and waiting for axis: ", axis)
mm.moveToPosition(axis, distance)
mm.waitForMotionCompletion(axis)

print("Done!")
```

getAxisMaxAcceleration(self, axis)

- **desc** Get maximum acceleration for a given axis.
  - **params**
    - **axis**
      - **desc** The target axis to read maximum acceleration from.
      - **type** Number of the AXIS_NUMBER class.
  - **compatibility** MachineMotion v2.
  - **returnValue** Max acceleration for the axis in mm/s^2.
  - **returnValueType** Number

  Example: speedAndAcceleration.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### speedAndAcceleration.py
# This example shows how to change the configured max speed and max acceleration for an axis,
# as well as how to retrieve the configured max speed, max acceleration and actual speed of a specific axis.

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

maxSpeedAxis = 50  # mm/s
maxAccelerationAxis = 150  # mm/s^2

# Setting speeds and accelerations

print("Setting max speed for axis ", axis, ": ", maxSpeedAxis, "mm/s")
mm.setAxisMaxSpeed(axis, maxSpeedAxis)
print("Setting max acceleration for axis ", axis, ": ", maxAccelerationAxis, "mm/s^2")
mm.setAxisMaxAcceleration(axis, maxAccelerationAxis)

# Getting speeds and accelerations

print("Getting speeds and accelerations...")

maxSpeed = mm.getAxisMaxSpeed(axis)
maxAccel = mm.getAxisMaxAcceleration(axis)
print(
    "For axis ",
    axis,
    " retrieved max speed: ",
    maxSpeed,
    "mm/s, max acceleration: ",
    maxAccel,
    "mm/s^2",
)

actualSpeed = mm.getActualSpeeds(axis)
print(
    "For axis ", axis, " retrieved actual speed: ", actualSpeed, "mm/s"
)  # The value should be 0 as the axis is not moving.

print("Getting actual speeds for all axes, in mm/s:")
actualSpeeds = mm.getActualSpeeds()
print(actualSpeeds)  # The values should be 0 as the axes are not moving.

print("Done!")
```

moveToPosition(self, axis, position)

- **desc** Moves the specified axis to a desired end location.
  - **params**
    - **axis**

- **desc** The axis which will perform the absolute move command.
- **type** Number

- **position**
  - **desc** The desired end position of the axis movement.
  - **type** Number

- **compatibility** MachineMotion v1 and MachineMotion v2.

Example: moveToPosition.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases moveToPosition with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1  # The axis that you'd like to move

# Movement configuration
position = 100  # The absolute position to which you'd like to move

# Home Axis before move to position
print("Axis " + str(axis) + " is going home.")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " homed.")

# Move
mm.moveToPosition(axis, position)
print("Axis " + str(axis) + " is moving towards position " + str(position) + "mm")
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " is at position " + str(position) + "mm")
```

moveToPositionCombined(self, axes, positions)

- **desc** Moves multiple specified axes to their desired end locations synchronously.
  - **params**
    - **axes**
      - **desc** The axes which will perform the move commands. Ex - [1 ,3]
      - **type** List
    - **positions**
      - **desc** The desired end position of all axess movement. Ex - [50, 10]
      - **type** List
  - **compatibility** MachineMotion v1 and MachineMotion v2.

- **note** The current speed and acceleration settings are applied to the combined motion of the axes.

Example: moveToPositionCombined.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases combined absolute moves with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 2
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 3
#   Drive Number(s): 3
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axesToMove = [AXIS_NUMBER.DRIVE1, AXIS_NUMBER.DRIVE2, AXIS_NUMBER.DRIVE3]

# Home actuators before performing absolute moves
print("Axes Moving Home Sequentially")
for axis in axesToMove:
    mm.moveToHome(axis)
    mm.waitForMotionCompletion()
print("Axes homed")

# Simultaneously moves three axis:
#   Moves axis 1 to absolute position 50mm
#   Moves axis 2 to absolute position 100mm
#   Moves axis 3 to absolute position 50mm
positions = [50, 100, 50]
mm.moveToPositionCombined(axesToMove, positions)
mm.waitForMotionCompletion()
for index, axis in enumerate(axesToMove):
    print("Axis " + str(axis) + " moved to position " + str(positions[index]) + "mm")
```

moveRelative(self, axis, distance)

- **desc** Moves the specified axis the specified distance.
  - **params**
    - **axis**
      - **desc** The axis to move.
      - **type** Integer
    - **distance**
      - **desc** The travel distance in mm.

- **type** Number

- **compatibility** MachineMotion v1 and MachineMotion v2.

Example: moveRelative.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases relative moves with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Begin Relative Move
distance = 100  # in mm
mm.moveRelative(axis, distance)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " moved " + str(distance) + "mm")

# Pass a negative distance value to move in the opposite direction
distance = -100  # in mm
mm.moveRelative(axis, distance)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " moved " + str(distance) + "mm")

print("--> Example Complete")
```

moveRelativeCombined(self, axes, distances)

- **desc** Moves the multiple specified axes the specified distances.
  - **params**
    - **axes**
      - **desc** The axes to move. Ex-[1,3]
      - **type** List of Integers
    - **distances**
      - **desc** The travel distances in mm. Ex - [10, 40]
      - **type** List of Numbers
  - **compatibility** MachineMotion v1 and MachineMotion v2.
  - **note** The current speed and acceleration settings are applied to the combined motion of the axes.

Example: moveRelativeCombined.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases combined relative moves with MachineMotion v1. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 2
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 3
#   Drive Number(s): 3
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axesToMove = [AXIS_NUMBER.DRIVE1, AXIS_NUMBER.DRIVE2, AXIS_NUMBER.DRIVE3]

# Simultaneously moves three axis:
#   Move axis 1 in the positive direction by 50 mm
#   Move axis 2 in the negative direction by 100 mm
#   Move axis 3 in the positive direction by 50 mm
distances = [50, 100, 50]
mm.moveRelativeCombined(axesToMove, distances)
mm.waitForMotionCompletion()
for index, axis in enumerate(axesToMove):
    print("Axis " + str(axis) + " moved " + str(distances[index]) + "mm")
```

setPosition(self, axis, position)

- **desc** Override the current position of the specified axis to a new value.
  - **params**
    - **axis**
      - **desc** Overrides the position on this axis.
      - **type** Number
    - **position**
      - **desc** The new position value in mm.
      - **type** Number
  - **compatibility** MachineMotion v1 and MachineMotion v2.

  Example: setPosition.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to set actuator position with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Home the actuator
print("Axis " + str(axis) + " will home")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " homed")

### Perform asolute moves with different reference points ###

print("Absolute Moves are referenced from home")
position = 100
mm.moveToPosition(axis, position)
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " is " + str(position) + "mm away from home.")

# Change the reference point to the current position
mm.setPosition(axis, 0)
print(
    "Absolute moves on axis "
    + str(axis)
    + " are now referenced from "
    + str(position)
    + "mm from home. "
)
time.sleep(2)

# Move again
position2 = 30
print(
    "Now moving to absolute position "
    + str(position2)
    + " mm, referenced from location 'setPosition' was called"
)
mm.moveToPosition(axis, position2)
mm.waitForMotionCompletion()
print(
    "Axis "
    + str(axis)
    + " is now "
    + str(position2)
    + "mm from reference position and "
    + str(position + position2)
    + "mm from home"
)
```

emitgCode(self, gCode)

- **desc** Executes raw gCode on the controller.
  - **params**
    - **gCode**
      - **desc** The g-code that will be passed directly to the controller.
      - **type** String
  - **note** All movement commands sent to the controller are by default in mm.
  - **compatibility** Recommended for MachineMotion v1.

isMotionCompleted(self, axis)

- **desc** Indicates if the last move command has completed.
  - **params**
    - **axis**
      - **desc** Target axis to check. If None is passed, all axes will be checked.
      - **type** Number of the AXIS_NUMBER class, or None
      - **default** None
  - **returnValue** Returns false if the machine is currently executing a movement command.
  - **returnValueType** Boolean or Dictionary of Booleans
  - **compatibility** MachineMotion v1 and MachineMotion v2.
  - **note** isMotionCompleted does not account for on-going continuous moves.

  Example: getPositions.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to read actuator positions with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 2, 3, 4
#   Any valid configuration

mm = MachineMotionV2()

# Configure actuator
axis = AXIS_NUMBER.DRIVE1

# Home Axis Before Moving
print("--> Axis " + str(axis) + " moving home")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
```

```
print("--> Axis " + str(axis) + " homed")


###### READ POSITIONS ######

# Read the position of one axis
print("--> Read the position of one axis")
actualPosition_axis = mm.getActualPositions(axis)
print(
    "Actual position of axis "
    + str(axis)
    + " is : "
    + str(actualPosition_axis)
    + " mm."
)

# Read the position of several axes
print("--> Read the position of several axes")
actualPositions = mm.getActualPositions()
print("Actual position of axis 1 is : " + str(actualPositions[1]) + " mm.")
print("Actual position of axis 2 is : " + str(actualPositions[2]) + " mm.")
print("Actual position of axis 3 is : " + str(actualPositions[3]) + " mm.")
print("Actual position of axis 4 is : " + str(actualPositions[4]) + " mm.")

###### MOVE AND READ POSITIONS ######

# Define Motion Parameters
distance = 100

# Move 100mm and check position again
mm.moveRelative(axis, distance)
print("--> Move ongoing")
while not mm.isMotionCompleted():
    actualPosition_axis = mm.getActualPositions(axis)
    print(
        "Actual position of axis "
        + str(axis)
        + " is : "
        + str(actualPosition_axis)
        + " mm."
    )

mm.waitForMotionCompletion()
print("--> Move completed")
actualPosition_axis = mm.getActualPositions(axis)
print(
    "Actual position of axis "
    + str(axis)
    + " is : "
    + str(actualPosition_axis)
    + " mm."
)

print("--> End of example")
```

waitForMotionCompletion(self, axis)

- **desc** Pauses python program execution until machine has finished its current movement.
    - **params**

- **axis** Target axis to check. If None is passed, all axes will be checked.
- **type** Number or None
- **default** None

- **compatibility** MachineMotion v1 and MachineMotion v2.
- **note** waitForMotionCompletion does not account for on-going continuous moves.

Example: waitForMotionCompletion.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to wait for actuator motion completion on MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Move the axis by 100mm
distance = 100

print("Moving %d mm!" % distance)
mm.moveRelative(axis, distance)
print("This message gets printed immediately")
mm.waitForMotionCompletion()
print("This message gets printed once machine has finished moving")
```

Example: moveIndependentAxis.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### This Python example configures and moves an independent axis.
### The individual axis speed and acceleration are set,
### then motion on the axis is selectively started, stopped and monitored

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large Servo

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Move, stop and monitor the axis
axisSpeed = 50  # mm/s
axisAcceleration = 50  # mm/s^2
distance = 250  # mm

print(
    "Configuring axis: ",
    axis,
    " speed: ",
    axisSpeed,
    "mm/s",
    " acceleration: ",
    axisAcceleration,
    "mm/s^2",
)
mm.setAxisMaxSpeed(axis, axisSpeed)
mm.setAxisMaxAcceleration(axis, axisAcceleration)

print("Relative move: ", distance, "mm")
mm.moveRelative(axis, distance)

axisMotionCompleted = mm.isMotionCompleted(axis)
if axisMotionCompleted:
    print("Axis ", axis, " is NOT currently moving.")
else:
    print("Axis ", axis, " is currently moving.")

time.sleep(3)

print("Stopping just axis: ", axis)
mm.stopAxis(axis)

time.sleep(3)

print("Moving and waiting for axis: ", axis)
mm.moveToPosition(axis, distance)
mm.waitForMotionCompletion(axis)

print("Done!")
```

configAxis(self, axis, uStep, mechGain)

- **desc** Configures motion parameters for a single axis on the MachineMotion v1.
  - **params**
    - **axis**
      - **desc** The axis to configure.
      - **type** Number
    - **uStep**
      - **desc** The microstep setting of the axis.
      - **type** Number
    - **mechGain**
      - **desc** The distance moved by the actuator for every full rotation of the stepper motor, in mm/revolution.
      - **type** Number
  - **note** The uStep setting is hardcoded into the machinemotion controller through a DIP switch and is by default set to 8. The value here must match the value on the DIP Switch.
  - **compatibility** MachineMotion v1 only.

configAxisDirection(self, axis, direction)

- **desc** Configures a single axis to operate in either clockwise (normal) or counterclockwise (reverse) mode. Refer to the Automation System Diagram for the correct axis setting.
  - **params**
    - **axis**
      - **desc** The specified axis.
      - **type** Number
    - **direction**
      - **desc** A string from the DIRECTION class. Either 'DIRECTION.NORMAL' or 'DIRECTION.REVERSE'. Normal direction means the axis will home towards end stop sensor A and reverse will make the axis home towards end stop B.
      - **type** String
  - **compatibility** MachineMotion v1 only.

configActuator(self, config, parentDrive)

- **desc** Configures motion parameters for a single or group of drives on a MachineMotion v2
  - **params**
    - **config**
      - **desc** The configuration to be applied to the drives
      - **type** Instance of the ActuatorConfigParams class

- **parentDrive**
  - **desc** The configuration of the parent drive of the axis being configured
  - **type** Instance of the DriveConfigParams class
- ***childDrives**
  - **desc** The configurations of the child drives in the axis being configured
  - **type** Instance or tuple of instances of the DriveConfigParam class (Maximum 3)

- **note** exactly one drive in *drives must be the parent.
- **compatibility** MachineMotion v2 only.

Example: configActuator.py

```
import sys
sys.path.append("../")
from MachineMotion import *

### This Python example configures actuators for a MachineMotion v2. ###
mm = MachineMotionV2()

# Configure a roller conveyor with a single drive with minimum parameters

# The following configuration is added
# Axis 1:
#   Drive Number(s): 1
#   Axis Type: Roller Conveyor
#   Motor Size: Large Servo
print("--> Configuring single drive roller conveyor.")
singleDefaultConfig = ActuatorConfigParams(
    AXIS_TYPE.ROLLER_CONVEYOR,
    MOTOR_SIZE.LARGE
)
singleDefaultDrive = DriveConfigParams(
    AXIS_NUMBER.DRIVE1,
    DIRECTION.NORMAL
)
mm.configActuator(singleDefaultConfig, singleDefaultDrive)

# Configure a CUSTOM actuator.

# The following configuration is added
# Axis 2:
#   Drive Number(s): 2
#   Axis Type: Custom
#   Motor Size: Medium Servo
#   Brake: None
#   Tuning Profile: Default
#   Motor Current: 10 A
#   Gain: 100 mm/turn
#   Control Loop: Closed Loop
print("--> Configuring custom axis.")
motorCurrent = 10  # A
mechGain = 100  # mm/turn

customConfig = ActuatorConfigParams(
    axisType=AXIS_TYPE.CUSTOM,
    motorSize=MOTOR_SIZE.MEDIUM,
    brake=BRAKE.NONE,
    tuningProfile=TUNING_PROFILES.DEFAULT,
    motorCurrent=motorCurrent,
    gain=mechGain,
    controlLoop=CONTROL_LOOPS.CLOSED_LOOP,
)
customDrive = DriveConfigParams(
```

```
    customDrive = DriveConfigParams(
        AXIS_NUMBER.DRIVE2,
        DIRECTION.REVERSE
    )
    mm.configActuator(customConfig, customDrive)

    # Configure a multi-drive timing belt.
    # The following configuration is added
    # Axis 3:
    #   Drive Number(s): 3, 4
    #   Axis Type: Timing Belt
    #   Motor Size: Small Servo
    #   Gearbox: 1/5 Gearbox
    print("--> Configuring multidrive timing belt.")
    multidriveConfig = ActuatorConfigParams(
        axisType=AXIS_TYPE.TIMING_BELT,
        motorSize=MOTOR_SIZE.SMALL,
        gearbox=GEARBOX.RATIO_5_TO_1,
    )
    multidriveParentDrive = DriveConfigParams(AXIS_NUMBER.DRIVE3, DIRECTION.NORMAL)
    multidriveChildDrive = DriveConfigParams(AXIS_NUMBER.DRIVE4, DIRECTION.REVERSE)
    mm.configActuator(multidriveConfig, multidriveParentDrive, multidriveChildDrive)

    print("--> Configuration complete!")
```

detectIOModules(self)

- **desc** Returns a dictionary containing all detected IO Modules.
    - **note** For more information, please see the digital IO datasheet here
    - **returnValue** Dictionary with keys of format “Digital IO Network Id [id]” and values [id] where [id] is the network IDs of all connected digital IO modules.
    - **returnValueType** Dictionary
    - **compatibility** MachineMotion v1 and MachineMotion v2.

    Example: digitalRead.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example reads digital inputs for MachineMotion v2. ###

mm = MachineMotionV2()

# Detect all connected digital IO Modules
detectedIOModules = mm.detectIOModules()

# Read and print all input values
if detectedIOModules is not None:
    for IO_Name, IO_NetworkID in detectedIOModules.items():
        readPins = [0, 1, 2, 3]
        for readPin in readPins:
            pinValue = mm.digitalRead(IO_NetworkID, readPin)
            print(
                "Pin " + str(readPin) + " on " + IO_Name + " has value " + str(pinValue)
            )
```

digitalRead(self, deviceNetworkId, pin)

- **desc** Reads the state of a digital IO modules input pins.
  - **params**
    - **deviceNetworkId**
      - **desc** The IO Module's device network ID. Defined by its onboard dipswitch.
      - **type** Integer between 1 and 8
    - **pin**
      - **desc** The index of the input pin.
      - **type** Integer
  - **returnValue** Returns 1 if the input pin is logic HIGH (24V) and returns 0 if the input pin is logic LOW (0V).
  - **compatibility** MachineMotion v1 and MachineMotion v2.

  Example: digitalRead.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example reads digital inputs for MachineMotion v2. ###

mm = MachineMotionV2()

# Detect all connected digital IO Modules
detectedIOModules = mm.detectIOModules()

# Read and print all input values
if detectedIOModules is not None:
    for IO_Name, IO_NetworkID in detectedIOModules.items():
        readPins = [0, 1, 2, 3]
        for readPin in readPins:
            pinValue = mm.digitalRead(IO_NetworkID, readPin)
            print(
                "Pin " + str(readPin) + " on " + IO_Name + " has value " + str(pinValue)
            )
```

digitalWrite(self, deviceNetworkId, pin, value)

- **desc** Sets voltage on specified pin of digital IO output pin to either logic HIGH (24V) or LOW (0V).
  - **params**
    - **deviceNetworkId**
      - **desc** The IO Module's device network ID. Defined by its onboard dipswitch.
      - **type** Integer
    - **pin**
      - **desc** The output pin number to write to.
      - **type** Integer
    - **value**
      - **desc** Writing '1' or HIGH will set digial output to 24V, writing 0 will set digital output to 0V.
      - **type** Integer
  - **compatibility** MachineMotion v1 and MachineMotion v2.
  - **note** Output pins maximum sourcing current is 75 mA and the maximum sinking current is 100 mA. On older digital IO modules, the pin labels on the digital IO module (pin 1, pin 2, pin 3, pin 4) correspond in software to (0, 1, 2, 3). Therefore, digitalWrite(deviceNetworkId, 2, 1) will set output pin 3 to 24V.

  Example: digitalWrite.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example writes digital outputs for MachineMotion v2. ###

mm = MachineMotionV2()

# Detect all connected digital IO Modules
detectedIOModules = mm.detectIOModules()

# Toggles the output pins
if detectedIOModules is not None:
    for IO_Name, IO_NetworkID in detectedIOModules.items():
        writePins = [0, 1, 2, 3]
        for writePin in writePins:
            print(
                "Pin " + str(writePin) + " on " + IO_Name + " is going to flash twice"
            )

            mm.digitalWrite(IO_NetworkID, writePin, 1)
            time.sleep(1)
            mm.digitalWrite(IO_NetworkID, writePin, 0)
            time.sleep(1)
            mm.digitalWrite(IO_NetworkID, writePin, 1)
            time.sleep(1)
            mm.digitalWrite(IO_NetworkID, writePin, 0)
            time.sleep(1)
```

setPowerSwitch(self, deviceNetworkId, switchState)

- **desc** Sets a switch of a power switch module to ON (closed) or OFF (open).
  - **params**
    - **deviceNetworkId**
      - **desc** Power switch device network ID. Defined by its onboard dipswitch.
      - **type** Integer
    - **value**
      - **desc** Writing POWER_SWITCH.ON will set the switch to closed, writing POWER_SWITCH.OFF will set the switch to open.
      - **type** Boolean or String of the POWER_SWITCH class
  - **compatibility** MachineMotion v2.

  Example: powerSwitch.py

```
import sys
import time

sys.path.append("../")
from MachineMotion import *

### This Python example control a power switch module on MachineMotion v2. ###

mm = MachineMotionV2()

deviceNetworkId = 7

### Set the power switch
print("--> Turning power switch on")
mm.setPowerSwitch(deviceNetworkId, POWER_SWITCH.ON)
time.sleep(3)
print("--> Turning power switch off")
mm.setPowerSwitch(deviceNetworkId, POWER_SWITCH.OFF)
time.sleep(3)

print("--> Example Complete")
```

waitOnPushButton(self, deviceNetworkId, button, state, timeout)

- **desc** Wait until a push button has reached a desired state
  - **params**
    - **deviceNetworkId**
      - **desc** The Push-Button module's device network ID. Defined by its onboard dipswitch.
      - **type** Integer
    - **button**
      - **desc** The address of the button (PUSH_BUTTON.COLOR.BLACK or PUSH_BUTTON.COLOR.WHITE) you want to readsee PUSH_BUTTON class
      - **type** Integer of the PUSH_BUTTON.COLOR class
    - **state**
      - **desc** The state of the push button (PUSH_BUTTON.STATE.PUSHED or PUSH_BUTTON.STATE.RELEASED) necessary to proceedsee PUSH_BUTTON class
      - **type** String of the PUSH_BUTTON.STATE class
      - **default** By default, this function will wait until the desired push button is PUSH_BUTTON.STATE.PUSHED.
    - **timeout**
      - **desc** The maximum time (seconds) the function will wait until it returns.If no timeout is passed, the function will wait indefinitely.
      - **type** Integer, Float or None
      - **default** None
  - **returnValue** True if push button has reached the desired state, False if the timeout has been reached.
  - **compatibility** MachineMotion v2.

Example: pushButton.py

```
import sys
import time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases different uses for a push button module on MachineMotion v2. ###
mm = MachineMotionV2()

### Define the module and button you would like to monitor.
deviceNetworkId = 5
blackButton = PUSH_BUTTON.COLOR.BLACK
whiteButton = PUSH_BUTTON.COLOR.WHITE

### Block the script until a button is pushed:
print("--> Waiting 5 seconds for the white button to be pushed!")
buttonWasPushed = mm.waitOnPushButton(
    deviceNetworkId, whiteButton, PUSH_BUTTON.STATE.PUSHED, 5
)
if buttonWasPushed:
    print("--> White button was pushed!")
else:
    print("--> waitOnPushButton timed out!")

time.sleep(1)

### Manually read the state of a push button
buttonState = mm.readPushButton(deviceNetworkId, whiteButton)
print("--> Reading from white push button: " + buttonState)
time.sleep(2)

### Define a callback to process push button status
def templateCallback(button_state):
    print(
        "--> templateCallback: Black push button on module: "
        + str(deviceNetworkId)
        + " has changed state."
    )
    if button_state == PUSH_BUTTON.STATE.PUSHED:
        print("--> Black push button has been pushed.")
    elif button_state == PUSH_BUTTON.STATE.RELEASED:
        print("--> Black push button has been released.")


###  You must first bind the callback function!
mm.bindPushButtonEvent(deviceNetworkId, blackButton, templateCallback)

print("--> Push the black button to trigger event! Press ctrl-c to exit")

while True:
    time.sleep(0.5)
```

bindPushButtonEvent(self, deviceNetworkId, button, callback_function)

- **desc** Configures a user-defined function to execute immediately after a change of state of a push button module button
  - **params**
    - **deviceNetworkId**

        - **desc** The Push-Button module's device network ID. Defined by its onboard dipswitch.
        - **type** Integer
    - **button**
        - **desc** The address of the button (PUSH_BUTTON.COLOR.BLACK or PUSH_BUTTON.COLOR.WHITE) you want to readsee PUSH_BUTTON class
        - **type** Integer of the PUSH_BUTTON.COLOR class
    - **callback_function**
        - **desc** The function to be executed after a push button changes state.
        - **type** function
- **note** The callback function used will be executed in a new thread.
- **compatibility** MachineMotion v2.

Example: pushButton.py

```
import sys
import time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases different uses for a push button module on MachineMotion v2. ###
mm = MachineMotionV2()

### Define the module and button you would like to monitor.
deviceNetworkId = 5
blackButton = PUSH_BUTTON.COLOR.BLACK
whiteButton = PUSH_BUTTON.COLOR.WHITE

### Block the script until a button is pushed:
print("--> Waiting 5 seconds for the white button to be pushed!")
buttonWasPushed = mm.waitOnPushButton(
    deviceNetworkId, whiteButton, PUSH_BUTTON.STATE.PUSHED, 5
)
if buttonWasPushed:
    print("--> White button was pushed!")
else:
    print("--> waitOnPushButton timed out!")

time.sleep(1)

### Manually read the state of a push button
buttonState = mm.readPushButton(deviceNetworkId, whiteButton)
print("--> Reading from white push button: " + buttonState)
time.sleep(2)

### Define a callback to process push button status
def templateCallback(button_state):
    print(
        "--> templateCallback: Black push button on module: "
        + str(deviceNetworkId)
        + " has changed state."
    )
    if button_state == PUSH_BUTTON.STATE.PUSHED:
        print("--> Black push button has been pushed.")
    elif button_state == PUSH_BUTTON.STATE.RELEASED:
        print("--> Black push button has been released.")


###  You must first bind the callback function!
mm.bindPushButtonEvent(deviceNetworkId, blackButton, templateCallback)

print("--> Push the black button to trigger event! Press ctrl-c to exit")

while True:
    time.sleep(0.5)
```

readPushButton(self, deviceNetworkId, button)

- **desc** Reads the state of a Push-Button module button.
  - **params**
    - **deviceNetworkId**

        - **desc** The Push-Button module's device network ID. Defined by its onboard dipswitch.
        - **type** Integer of the PUSH_BUTTON.COLOR class
    - **button**
        - **desc** The address of the button (PUSH_BUTTON.COLOR.BLACK or PUSH_BUTTON.COLOR.WHITE) you want to readsee PUSH_BUTTON class
        - **type** Integer of the PUSH_BUTTON.COLOR class
- **returnValue** Returns PUSH_BUTTON.STATE.RELEASED if the input button is released and returns PUSH_BUTTON.STATE.PUSHED if the input button pushed
- **compatibility** MachineMotion v2.

Example: pushButton.py

```
import sys
import time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases different uses for a push button module on MachineMotion v2. ###
mm = MachineMotionV2()

### Define the module and button you would like to monitor.
deviceNetworkId = 5
blackButton = PUSH_BUTTON.COLOR.BLACK
whiteButton = PUSH_BUTTON.COLOR.WHITE

### Block the script until a button is pushed:
print("--> Waiting 5 seconds for the white button to be pushed!")
buttonWasPushed = mm.waitOnPushButton(
    deviceNetworkId, whiteButton, PUSH_BUTTON.STATE.PUSHED, 5
)
if buttonWasPushed:
    print("--> White button was pushed!")
else:
    print("--> waitOnPushButton timed out!")

time.sleep(1)

### Manually read the state of a push button
buttonState = mm.readPushButton(deviceNetworkId, whiteButton)
print("--> Reading from white push button: " + buttonState)
time.sleep(2)

### Define a callback to process push button status
def templateCallback(button_state):
    print(
        "--> templateCallback: Black push button on module: "
        + str(deviceNetworkId)
        + " has changed state."
    )
    if button_state == PUSH_BUTTON.STATE.PUSHED:
        print("--> Black push button has been pushed.")
    elif button_state == PUSH_BUTTON.STATE.RELEASED:
        print("--> Black push button has been released.")


###  You must first bind the callback function!
mm.bindPushButtonEvent(deviceNetworkId, blackButton, templateCallback)

print("--> Push the black button to trigger event! Press ctrl-c to exit")

while True:
    time.sleep(0.5)
```

readEncoder(self, encoder, readingType)

- **desc** Returns the last received encoder position in counts.
  - **params**
    - **encoder**

        - **desc** The identifier of the encoder to read
        - **type** Integer
    - **readingType**
        - **desc** Either 'real time' or 'stable'. In 'real time' mode, readEncoder will return the most recently received encoder information. In 'stable' mode, readEncoder will update its return value only after the encoder output has stabilized around a specific value, such as when the axis has stopped motion.
        - **type** String
- **returnValue** The current position of the encoder, in counts. The encoder has 3600 counts per revolution.
- **returnValueType** Integer
- **compatibility** MachineMotion v1 only.
- **note** The encoder position returned by this function may be delayed by up to 250 ms due to internal propogation delays.

bindeStopEvent(self, callback_function)

- **desc** Configures a user defined function to execute immediately after an E-stop event.
    - **params**
        - **callback_function**
            - **type** function
            - **desc** The function to be executed after an e-stop is triggered or released.
    - **compatibility** MachineMotion v1 and MachineMotion v2.

    Example: eStop.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how monitor eStop on MachineMotion v2. ###

# Create MachineMotion instance
mm = MachineMotionV2()
time.sleep(0.1)  # Wait to initialize internal eStop topics.

# Define a callback to process estop status
def handleEstopStatus(status):
    if status == True:
        # executed when you enter estop
        print("MachineMotion is in estop!")
        print("Estop must be reset with a physical module")
    if status == False:
        # executed when you exit estop
        print("MachineMotion not in estop...")
        print("Estop must be triggered with a physical module")


mm.bindeStopEvent(handleEstopStatus)

while True:
    time.sleep(1)
```

lockBrake(self, aux_port_number, safety_adapter_presence)

- **desc** Lock the brake, by shutting off the power of the designated AUX port of the MachineMotion (0V).
  - **params**
    - **aux_port_number**
      - **type** Integer
      - **desc** The number of the AUX port the brake is connected to.
    - **safety_adapter_presence**
      - **type** Boolean
      - **desc** Is a yellow safety adapter plugged in between the brake cable and the AUX port.
  - **compatibility** MachineMotion v1 and MachineMotion v2.
  - **note** This function is compatible only with V1F and more recent MachineMotions.

  Example: controlBrakes.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example control brakes on MachineMotion v2. ###

mm = MachineMotionV2()

# Define the brake parameters
axis = 1  # Drive number
safety_adapter_presence = True  # Is a yellow safety adapter plugged in between the brake cable and the brake port
# Important Note : This flag must always be set to "True" on MachineMotions v2.

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)

# Lock the brake
mm.lockBrake(axis, safety_adapter_presence)
time.sleep(0.2)
# Wait before reading brake state, for it to get correctly updated

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)

# DO NOT MOVE WHILE BRAKE IS ENGAGED
print("Waiting two seconds. Do not move the actuator while the brake is locked !")
time.sleep(2)  # Unlock the brake after two seconds

# Release the brakes
mm.unlockBrake(axis, safety_adapter_presence)
time.sleep(0.2)
# Wait before reading brake state, for it to get correctly updated

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)
```

unlockBrake(self, aux_port_number, safety_adapter_presence)

- **desc** Unlock the brake, by powering on the designated AUX port of the MachineMotion (24V).
  - **params**
    - **aux_port_number**
      - **type** Integer
      - **desc** The number of the AUX port the brake is connected to.
    - **safety_adapter_presence**
      - **type** Boolean
      - **desc** Is a yellow safety adapter plugged in between the brake cable and the AUX port.
  - **compatibility** MachineMotion v1 and MachineMotion v2.
  - **note** This function is compatible only with V1F and more recent MachineMotions.

Example: controlBrakes.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example control brakes on MachineMotion v2. ###

mm = MachineMotionV2()

# Define the brake parameters
axis = 1  # Drive number
safety_adapter_presence = True  # Is a yellow safety adapter plugged in between the brake cable and the brake port
# Important Note : This flag must always be set to "True" on MachineMotions v2.

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)

# Lock the brake
mm.lockBrake(axis, safety_adapter_presence)
time.sleep(0.2)
# Wait before reading brake state, for it to get correctly updated

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)

# DO NOT MOVE WHILE BRAKE IS ENGAGED
print("Waiting two seconds. Do not move the actuator while the brake is locked !")
time.sleep(2)  # Unlock the brake after two seconds

# Release the brakes
mm.unlockBrake(axis, safety_adapter_presence)
time.sleep(0.2)
# Wait before reading brake state, for it to get correctly updated

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)
```

getBrakeState(self, aux_port_number, safety_adapter_presence)

- **desc** Read the current state of the brake connected to a given AUX port of the MachineMotion.
  - **params**
    - **aux_port_number**
      - **type** Integer
      - **desc** The number of the AUX port the brake is connected to.
    - **safety_adapter_presence**
      - **type** Boolean
      - **desc** Is a yellow safety adapter plugged in between the brake cable and the AUX port.
  - **returnValue** The current state of the brake, as determined according to the current voltage of the AUX port (0V or 24V). The returned String can be “locked”, “unlocked”, or “unknown” (for MachineMotions prior to the V1F hardware version), as defined by the BRAKE_STATES class.
  - **returnValueType** String
  - **compatibility** MachineMotion v1 and MachineMotion v2.
  - **note** This function is compatible only with V1F and more recent MachineMotions.

  Example: controlBrakes.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example control brakes on MachineMotion v2. ###

mm = MachineMotionV2()

# Define the brake parameters
axis = 1  # Drive number
safety_adapter_presence = True  # Is a yellow safety adapter plugged in between the brake cable and the brake port
# Important Note : This flag must always be set to "True" on MachineMotions v2.

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)

# Lock the brake
mm.lockBrake(axis, safety_adapter_presence)
time.sleep(0.2)
# Wait before reading brake state, for it to get correctly updated

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)

# DO NOT MOVE WHILE BRAKE IS ENGAGED
print("Waiting two seconds. Do not move the actuator while the brake is locked !")
time.sleep(2)  # Unlock the brake after two seconds

# Release the brakes
mm.unlockBrake(axis, safety_adapter_presence)
time.sleep(0.2)
# Wait before reading brake state, for it to get correctly updated

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)
```

startPath(self, path, maxAcceleration, speedOverride, rapidSpeed)

- **desc** Loads and starts a G-Code path. Path mode must first be configured (see configPathMode).
  - **params**
    - **path**
      - **desc** G-Code string
      - **type** string
    - **maxAcceleration**
      - **desc** Defines the maximum system acceleration on the path.
      - **type** number
    - **speedOverride**
      - **desc** Optional.Allows for slowing down or speeding up a G-Code path. When parsed, the path speed will be multiplied by this value.
      - **type** positive number

- **default** 1.0

- **rapidSpeed**
  - **desc** Optional. Allows to manually set the rapid speed in mm/s for G0 commands. If none, rapid speed will default to the lowest axis max speed.
  - **type** positive number
  - **default** None

- **compatibility** MachineMotion v2.

Example: pathFollowing.py

```
import sys
sys.path.append("../")
from MachineMotion import *

mm = MachineMotionV2()

### This example demonstrates the creation and execution of a simple 2D path ###
# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1, 2
#   Axis Type: Enclosed Timing Belt with 5:1 gearbox
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 3
#   Axis Type: Enclosed Ballscrew
#   Motor Size: Large

GCODE_STRING = """
(Operational mode commands)
G90   ; Absolute position mode
G90.1 ; Absolute arc centre
G21   ; Use millimeter units
G17   ; XY plane arcs
G64 P0.5 ; Blend move mode, 0.5 mm tolerance

(Movement and output commands)
G0 X60 Y110 F1200 ; Rapid move at 1200 mm/minute
M3 $1 ; Start tool 1 in clockwise direction
G1 X110 Y110 F1000 ; Travel move at 1000 mm/minute
G2 X110 Y10 I100 J60 ; Clockwise arc
G1 X60 Y10
G2 X60 Y110 I60 J60
G4 P1.0 ; Dwell for 1 second
M5 $1 ; Stop tool 1
G0 X1 Y1
"""

# Optional: If your path contains M3-4-5 commands,
# define device,port and on value of both tool directions.
# clockwise must be defined for M3 commands
clockwise = DIGITAL_OUTPUT_STATE(deviceId=1, port=0, value=1)
counterClockwise = DIGITAL_OUTPUT_STATE(1, 1, 1)
tool1 = TOOL(1, clockwise, counterClockwise)
# associate with the parent drive
driveAssociation = {"X": 1, "Y": 3}
# Configure the path following mode
mm.configPathMode(driveAssociation, tool1)
maxPathAcceleration = 1000  # mm/s^2

mm.setAxisMaxAcceleration(1,1000)
mm.setAxisMaxAcceleration(3,1000)
mm.setAxisMaxSpeed(1,500)
mm.setAxisMaxSpeed(3,500)
```

```
mm.moveToHome(1)
mm.moveToHome(3)
mm.waitForMotionCompletion()

# Start Path Following.
mm.startPath(GCODE_STRING, maxPathAcceleration)
running = True
while running:
    pathStatus = mm.getPathStatus()
    running = pathStatus['running']
    print('--> Got Path Status: ', pathStatus)
    time.sleep(0.5)

mm.stopPath()
print("--> Done!")
```

configPathMode(self, axisMap)

- **desc** Each G-Code Path controls axes (e.g. X,Y,Z). Some paths control tools (via T, M3,4,5,6 commands).configPathMode Configures relevant axes and digital outputs for path following.
  - **params**
    - **axisMap**
      - **desc** A mapping between each G-Code axis and it associated parent drive, e.g. {“X”: 1, “Y”: 2, “Z”: 3}.
      - **type** Dict of axis string keys (AXIS_NAME class) and integer drive numbers (AXIS_NUMBER class)
    - ***tools**
      - **desc** Tuple of all the tools that will be used in the G-Code path.
      - **type** tool or tuple of tools of the TOOL class
  - **compatibility** MachineMotion v2

  Example: pathFollowing.py

```
import sys
sys.path.append("../")
from MachineMotion import *

mm = MachineMotionV2()

### This example demonstrates the creation and execution of a simple 2D path ###
# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1, 2
#   Axis Type: Enclosed Timing Belt with 5:1 gearbox
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 3
#   Axis Type: Enclosed Ballscrew
#   Motor Size: Large

GCODE_STRING = """
(Operational mode commands)
G90   ; Absolute position mode
G90.1 ; Absolute arc centre
G21   ; Use millimeter units
G17   ; XY plane arcs
```

```
G64 P0.5 ; Blend move mode, 0.5 mm tolerance

(Movement and output commands)
G0 X60 Y110 F1200 ; Rapid move at 1200 mm/minute
M3 $1 ; Start tool 1 in clockwise direction
G1 X110 Y110 F1000 ; Travel move at 1000 mm/minute
G2 X110 Y10 I100 J60 ; Clockwise arc
G1 X60 Y10
G2 X60 Y110 I60 J60
G4 P1.0 ; Dwell for 1 second
M5 $1 ; Stop tool 1
G0 X1 Y1
"""

# Optional: If your path contains M3-4-5 commands,
# define device,port and on value of both tool directions.
# clockwise must be defined for M3 commands
clockwise = DIGITAL_OUTPUT_STATE(deviceId=1, port=0, value=1)
counterClockwise = DIGITAL_OUTPUT_STATE(1, 1, 1)
tool1 = TOOL(1, clockwise, counterClockwise)
# associate with the parent drive
driveAssociation = {"X": 1, "Y": 3}
# Configure the path following mode
mm.configPathMode(driveAssociation, tool1)
maxPathAcceleration = 1000  # mm/s^2

mm.setAxisMaxAcceleration(1,1000)
mm.setAxisMaxAcceleration(3,1000)
mm.setAxisMaxSpeed(1,500)
mm.setAxisMaxSpeed(3,500)

mm.moveToHome(1)
mm.moveToHome(3)
mm.waitForMotionCompletion()

# Start Path Following.
mm.startPath(GCODE_STRING, maxPathAcceleration)
running = True
while running:
    pathStatus = mm.getPathStatus()
    running = pathStatus['running']
    print('--> Got Path Status: ', pathStatus)
    time.sleep(0.5)

mm.stopPath()
print("--> Done!")
```

getPathStatus(self)

- **desc** Get the live status of the path following procedue.
  - **returnValue** object
    - **“running”** boolean
    - **“line”** number,
    - **“command”** string,
    - **“speed”** number,
    - **“acceleration”** number,

- **"tool"** object,
- **"error"** string,

- **returnValueType** object
- **compatibility** MachineMotion v2

Example: pathFollowing.py

```
import sys
sys.path.append("../")
from MachineMotion import *

mm = MachineMotionV2()

### This example demonstrates the creation and execution of a simple 2D path ###
# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1, 2
#   Axis Type: Enclosed Timing Belt with 5:1 gearbox
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 3
#   Axis Type: Enclosed Ballscrew
#   Motor Size: Large

GCODE_STRING = """
(Operational mode commands)
G90   ; Absolute position mode
G90.1 ; Absolute arc centre
G21   ; Use millimeter units
G17   ; XY plane arcs
G64 P0.5 ; Blend move mode, 0.5 mm tolerance

(Movement and output commands)
G0 X60 Y110 F1200 ; Rapid move at 1200 mm/minute
M3 $1 ; Start tool 1 in clockwise direction
G1 X110 Y110 F1000 ; Travel move at 1000 mm/minute
G2 X110 Y10 I100 J60 ; Clockwise arc
G1 X60 Y10
G2 X60 Y110 I60 J60
G4 P1.0 ; Dwell for 1 second
M5 $1 ; Stop tool 1
G0 X1 Y1
"""

# Optional: If your path contains M3-4-5 commands,
# define device,port and on value of both tool directions.
# clockwise must be defined for M3 commands
clockwise = DIGITAL_OUTPUT_STATE(deviceId=1, port=0, value=1)
counterClockwise = DIGITAL_OUTPUT_STATE(1, 1, 1)
tool1 = TOOL(1, clockwise, counterClockwise)
# associate with the parent drive
driveAssociation = {"X": 1, "Y": 3}
# Configure the path following mode
mm.configPathMode(driveAssociation, tool1)
maxPathAcceleration = 1000  # mm/s^2

mm.setAxisMaxAcceleration(1,1000)
mm.setAxisMaxAcceleration(3,1000)
mm.setAxisMaxSpeed(1,500)
mm.setAxisMaxSpeed(3,500)

mm.moveToHome(1)
mm.moveToHome(3)
mm.waitForMotionCompletion()
```

```
# Start Path Following.
mm.startPath(GCODE_STRING, maxPathAcceleration)
running = True
while running:
    pathStatus = mm.getPathStatus()
    running = pathStatus['running']
    print('--> Got Path Status: ', pathStatus)
    time.sleep(0.5)

mm.stopPath()
print("--> Done!")
```

stopPath(self)

- **desc** Abruptly stop the path following procedure.
    - **compatibility** MachineMotion v2.

    Example: pathFollowing.py

```
import sys
sys.path.append("../")
from MachineMotion import *

mm = MachineMotionV2()

### This example demonstrates the creation and execution of a simple 2D path ###
# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1, 2
#   Axis Type: Enclosed Timing Belt with 5:1 gearbox
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 3
#   Axis Type: Enclosed Ballscrew
#   Motor Size: Large

GCODE_STRING = """
(Operational mode commands)
G90   ; Absolute position mode
G90.1 ; Absolute arc centre
G21   ; Use millimeter units
G17   ; XY plane arcs
G64 P0.5 ; Blend move mode, 0.5 mm tolerance

(Movement and output commands)
G0 X60 Y110 F1200 ; Rapid move at 1200 mm/minute
M3 $1 ; Start tool 1 in clockwise direction
G1 X110 Y110 F1000 ; Travel move at 1000 mm/minute
G2 X110 Y10 I100 J60 ; Clockwise arc
G1 X60 Y10
G2 X60 Y110 I60 J60
G4 P1.0 ; Dwell for 1 second
M5 $1 ; Stop tool 1
G0 X1 Y1
"""

# Optional: If your path contains M3-4-5 commands,
# define device,port and on value of both tool directions.
# clockwise must be defined for M3 commands
```

```
clockwise = DIGITAL_OUTPUT_STATE(deviceId=1, port=0, value=1)
counterClockwise = DIGITAL_OUTPUT_STATE(1, 1, 1)
tool1 = TOOL(1, clockwise, counterClockwise)
# associate with the parent drive
driveAssociation = {"X": 1, "Y": 3}
# Configure the path following mode
mm.configPathMode(driveAssociation, tool1)
maxPathAcceleration = 1000  # mm/s^2

mm.setAxisMaxAcceleration(1,1000)
mm.setAxisMaxAcceleration(3,1000)
mm.setAxisMaxSpeed(1,500)
mm.setAxisMaxSpeed(3,500)

mm.moveToHome(1)
mm.moveToHome(3)
mm.waitForMotionCompletion()

# Start Path Following.
mm.startPath(GCODE_STRING, maxPathAcceleration)
running = True
while running:
    pathStatus = mm.getPathStatus()
    running = pathStatus['running']
    print('--> Got Path Status: ', pathStatus)
    time.sleep(0.5)

mm.stopPath()
print("--> Done!")
```

onDigitalInput(self, callback)

- **desc** Set a callback function that is called when digital input updates are received. For internal use.
- **params**
  - **callback**
    - **desc** Function called when input value is updated.
    - **type** Callable[deviceType, deviceNetworkId, pin, value] -> None
- **returnValue** None
- **compatibility** MachineMotion v1 and MachineMotion v2.

configStepper(self, drive, mechGain, direction, motorCurrent, microSteps, motorSize)

- **desc** Configures motion parameters as a stepper motor, for a single drive on the MachineMotion v2.
- **params**
  - **drive**
    - **desc** The drive to configure.
    - **type** Number of the AXIS_NUMBER class
  - **mechGain**

        - **desc** The distance moved by the actuator for every full rotation of the stepper motor, in mm/revolution.
        - **type** Number of the MECH_GAIN class
    - **direction**
        - **desc** The direction of the axis
        - **type** String of the DIRECTION class
    - **motorCurrent**
        - **desc** The current to power the motor with, in Amps.
        - **type** Number
    - **microSteps**
        - **desc** The microstep setting of the drive.
        - **type** Number from MICRO_STEPS class
    - **motorSize**
        - **desc** The size of the motor(s) connected to the specified drive(s)
        - **type** String from the MOTOR_SIZE class
        - **default** MOTOR_SIZE.LARGE
- **note** Warning, changing the configuration can de-energize motors and thus cause unintended behaviour on vertical axes.
- **compatibility** MachineMotion v2 only.

configServo(self, drives, mechGain, directions, motorCurrent, tuningProfile, parentDrive, motorSize, brake)

- **desc** Configures motion parameters as a servo motor, for a single drive on the MachineMotion v2.
- **params**
    - **drives**
        - **desc** The drive or list of drives to configure.
        - **type** Number or list of numbers of the AXIS_NUMBER class
    - **mechGain**
        - **desc** The distance moved by the actuator for every full rotation of the stepper motor, in mm/revolution.
        - **type** Number of the MECH_GAIN class
    - **directions**
        - **desc** The direction or list of directions of each configured axis
        - **type** String or list of strings of the DIRECTION class. Must have the same length as drives
    - **motorCurrent**
        - **desc** The current to power the motor with, in Amps.
        - **type** Number
    - **tuningProfile**
        - **desc** The tuning profile of the smartDrive. Determines the characteristics of the servo motor's PID controller
        - **type** String of the TUNING_PROFILES class
        - **default** TUNING_PROFILES.DEFAULT

- **parentDrive**
  - **desc** The parent drive of the multi-drive axis. The axis' home and end sensors must be connected to this drive.
  - **type** Number
  - **default** None
- **motorSize**
  - **desc** The size of the motor(s) connected to the specified drive(s)
  - **type** String from the MOTOR_SIZE class
  - **default** MOTOR_SIZE.LARGE
- **brake**
  - **desc** The presence of a brake on the desired axis.If set to BRAKE.PRESENT, moving the axis while the brake is locked will generate an error.If set to BRAKE.NONE, motion will be allowed regardless of internal brake control.
  - **type** String of the BRAKE class
  - **default** BRAKE.NONE

- **note** Warning, changing the configuration can de-energize motors and thus cause unintended behaviour on vertical axes.
- **compatibility** MachineMotion v2 only.

setSpeed(self, speed, units)

- **desc** Sets the global speed for all movement commands on all axes.
- **params**
  - **speed**
    - **desc** The global max speed in mm/s, or mm/min according to the units parameter.
    - **type** Number
  - **units**
    - **desc** Units for speed. Can be switched to UNITS_SPEED.mm_per_min
    - **defaultValue** UNITS_SPEED.mm_per_sec
    - **type** String
- **compatibility** MachineMotion v1 and MachineMotion v2.

setAcceleration(self, acceleration, units)

- **desc** Sets the global acceleration for all movement commands on all axes.
- **params**
  - **mm_per_sec_sqr**
    - **desc** The global acceleration in mm/s^2.
    - **type** Number
  - **units**
    - **desc** Units for speed. Can be switched to UNITS_ACCEL.mm_per_min_sqr

- **defaultValue** UNITS_ACCEL.mm_per_sec_sqr
- **type** String

- **compatibility** MachineMotion v1 and MachineMotion v2.

class AXIS_TYPE

Variables

```
BALL_SCREW = "ball_screw"
BELT_CONVEYOR = "belt_conveyor"
BELT_RACK = "belt_rack"
ELECTRIC_CYLINDER = "electric_cylinder"
ENCLOSED_BALL_SCREW = "enclosed_ball_screw"
ENCLOSED_LEAD_SCREW = "enclosed_lead_screw"
ENCLOSED_TIMING_BELT = "enclosed_timing_belt"
HEAVY_DUTY_ROLLER_CONVEYOR = "heavy_duty_roller_conveyor"
INDEXER_V2 = "indexer_v2"
RACK_AND_PINION = "rack_and_pinion"
RACK_AND_PINION_V2 = "rack_and_pinion_v2"
ROLLER_CONVEYOR = "roller_conveyor"
TIMING_BELT = "belt"
TIMING_BELT_CONVEYOR = "timing_belt_conveyor"
TELESCOPIC_COLUMN = "telescopic_column"
CUSTOM = "custom"
```

class MACHINEMOTION_HW_VERSIONS

Variables

```
MMv1 = 1
MMv2 = 2
MMv2OneDrive = 3
```

class DIRECTION

Variables

```
POSITIVE = "positive"
NEGATIVE = "negative"
NORMAL = POSITIVE
REVERSE = NEGATIVE
CLOCKWISE = POSITIVE
COUNTERCLOCKWISE = NEGATIVE
```

class AXIS_NUMBER

Variables

```
DRIVE1 = 1
DRIVE2 = 2
DRIVE3 = 3
DRIVE4 = 4
```

class AXIS_NAME

Variables

```
X = "X"
Y = "Y"
Z = "Z"
W = "W"
```

class UNITS_SPEED

Variables

```
mm_per_min = "mm per minute"
mm_per_sec = "mm per second"
```

class UNITS_ACCEL

Variables

```
mm_per_min_sqr = "mm per minute"
mm_per_sec_sqr = "mm per second"
```

class DEFAULT_IP_ADDRESS

Variables

```
usb_windows = "192.168.7.2"
usb_mac_linux = "192.168.7.2"
ethernet = "192.168.0.2"
localhost = "127.0.0.1"
```

class MICRO_STEPS

Variables

```
ustep_full = 1
ustep_2 = 2
ustep_4 = 4
ustep_8 = 8
ustep_16 = 16
ustep_32 = 32
```

class MECH_GAIN

Variables

```
timing_belt_150mm_turn = 150
legacy_timing_belt_200_mm_turn = 200
enclosed_timing_belt_mm_turn = 208
ballscrew_10mm_turn = 10
enclosed_ballscrew_16mm_turn = 16
legacy_ballscrew_5_mm_turn = 5
indexer_deg_turn = 85
indexer_v2_deg_turn = 36
roller_conveyor_mm_turn = 157.7
belt_conveyor_mm_turn = 73.563
rack_pinion_mm_turn = 157.08
rack_pinion_v2_mm_turn = 141.37
electric_cylinder_mm_turn = 6
belt_rack_mm_turn = 125
enclosed_lead_screw_mm_turn = 4
hd_roller_conveyor_mm_turn = 123.3
```

class STEPPER_MOTOR

Variables

```
steps_per_turn = 200
```

class AUX_PORTS

Variables

```
aux_1 = 0
aux_2 = 1
aux_3 = 2
aux_4 = 3
```

class ENCODER_TYPE

Variables

```
real_time = "realtime-position"
stable = "stable-position"
```

class BRAKE_STATES

Variables

```
locked = "locked"
unlocked = "unlocked"
unknown = "unknown"
```

class BRAKE

Variables

```
PRESENT = "present"
NONE = "none"
```

class TUNING_PROFILES

Variables

```
DEFAULT = "default"
CONVEYOR_TURNTABLE = "conveyor_turntable"
CUSTOM = "custom"
```

class CONTROL_LOOPS

Variables

```
OPEN_LOOP = "open"
CLOSED_LOOP = "closed"
```

class POWER_SWITCH

Variables

```
ON = "on"
OFF = "off"
```

class PUSH_BUTTON

class PUSH_BUTTON.COLOR

Variables

```
BLACK = 0
WHITE = 1
```

class PUSH_BUTTON.STATE

Variables

```
PUSHED = "pushed"
RELEASED = "released"
```

class MOTOR_SIZE

Variables

```
SMALL = "Small Servo"
MEDIUM = "Medium Servo"
LARGE = "Large Servo"
ELECTRIC_CYLINDER = "Nema 17 Stepper"
```

class GEARBOX

Variables

```
RATIO_5_TO_1 = 1.0 / 5.0
NONE = 1.0
```

class MQTT

Variables

```
TIMEOUT = 10.0
```

class MQTT.PATH

Variables

```
ESTOP = "estop"
ESTOP_STATUS = ESTOP + "/status"
ESTOP_TRIGGER_REQUEST = ESTOP + "/trigger/request"
ESTOP_TRIGGER_RESPONSE = ESTOP + "/trigger/response"
ESTOP_RELEASE_REQUEST = ESTOP + "/release/request"
ESTOP_RELEASE_RESPONSE = ESTOP + "/release/response"
ESTOP_SYSTEMRESET_REQUEST = ESTOP + "/systemreset/request"
ESTOP_SYSTEMRESET_RESPONSE = ESTOP + "/systemreset/response"
AUX_PORT_POWER = "aux_power"
AUX_PORT_SAFETY = "aux_safety_power"
SMARTDRIVES_READY = "smartDrives/areReady"
```

class IO_STATE(object)

class DIGITAL_OUTPUT_STATE(IO_STATE)

__init__(self, deviceId, port, value)

- **desc** For use with path following applications.Defines a digital output's state, which can then be used to configure a connected peripheral's behaviour.
  - **params**
    - **deviceId**
      - **desc** The device ID of the module. Defined by its onboard dipswitch.
      - **type** Integer between 1 and 8
    - **port**
      - **desc** The output port of the module.
      - **type** integer between 0 and 3
    - **value**
      - **desc** The desired value being defined. 1 if the input pin is logic HIGH (24V), 0 if the input pin is logic LOW (0V).
      - **type** Integer. 0 or 1.

Example: pathFollowing.py

```
import sys
sys.path.append("../")
from MachineMotion import *

mm = MachineMotionV2()

### This example demonstrates the creation and execution of a simple 2D path ###
# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1, 2
#   Axis Type: Enclosed Timing Belt with 5:1 gearbox
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 3
#   Axis Type: Enclosed Ballscrew
#   Motor Size: Large

GCODE_STRING = """
(Operational mode commands)
G90   ; Absolute position mode
G90.1 ; Absolute arc centre
G21   ; Use millimeter units
G17   ; XY plane arcs
G64 P0.5 ; Blend move mode, 0.5 mm tolerance

(Movement and output commands)
G0 X60 Y110 F1200 ; Rapid move at 1200 mm/minute
M3 $1 ; Start tool 1 in clockwise direction
G1 X110 Y110 F1000 ; Travel move at 1000 mm/minute
G2 X110 Y10 I100 J60 ; Clockwise arc
G1 X60 Y10
G2 X60 Y110 I60 J60
G4 P1.0 ; Dwell for 1 second
M5 $1 ; Stop tool 1
G0 X1 Y1
"""

# Optional: If your path contains M3-4-5 commands,
# define device,port and on value of both tool directions.
# clockwise must be defined for M3 commands
clockwise = DIGITAL_OUTPUT_STATE(deviceId=1, port=0, value=1)
counterClockwise = DIGITAL_OUTPUT_STATE(1, 1, 1)
tool1 = TOOL(1, clockwise, counterClockwise)
```

```
# associate with the parent drive
driveAssociation = {"X": 1, "Y": 3}
# Configure the path following mode
mm.configPathMode(driveAssociation, tool1)
maxPathAcceleration = 1000  # mm/s^2

mm.setAxisMaxAcceleration(1,1000)
mm.setAxisMaxAcceleration(3,1000)
mm.setAxisMaxSpeed(1,500)
mm.setAxisMaxSpeed(3,500)

mm.moveToHome(1)
mm.moveToHome(3)
mm.waitForMotionCompletion()

# Start Path Following.
mm.startPath(GCODE_STRING, maxPathAcceleration)
running = True
while running:
    pathStatus = mm.getPathStatus()
    running = pathStatus['running']
    print('--> Got Path Status: ', pathStatus)
    time.sleep(0.5)

mm.stopPath()
print("--> Done!")
```

class TOOL

__init__(self, number, cwlo, ccwlo)

- **desc** A TOOL object defines a group of digital outputs to be used in path following applications.Each tool must have a clockwise direction, and may optionally have a counter-clockwise direction. When defined and configured,
  - **the T,M3,M4,M5 and M6 G-Code commands can be used (see** configPathMode)
  - **params**
    - **number**
      - **desc** Corresponds to the tool's index in the motion server's internal tool table.When defined, the desired tool can selected using the T command (e.g T1 in G-code selects tool 1).
      - **type** positive integer.
    - **cwlo**
      - **desc** The digital output mapping of device ID, port and value which will turn desired physical tool on in the clockwise direction.Related to the M3 and M5 commands in G-Code.
      - **type** Instance of the DIGITAL_OUTPUT_STATE class
    - **ccwlo**
      - **desc** Optional. The digital output mapping of device ID, port and value which will turn desired physical tool on in the counter-clockwise direction.Related to the M4 and M5 commands in G-Code.
      - **type** Instance of the DIGITAL_OUTPUT_STATE class

Example: pathFollowing.py

```
import sys
sys.path.append("../")
```

```
from MachineMotion import *

mm = MachineMotionV2()

### This example demonstrates the creation and execution of a simple 2D path ###
# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1, 2
#   Axis Type: Enclosed Timing Belt with 5:1 gearbox
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 3
#   Axis Type: Enclosed Ballscrew
#   Motor Size: Large

GCODE_STRING = """
(Operational mode commands)
G90   ; Absolute position mode
G90.1 ; Absolute arc centre
G21   ; Use millimeter units
G17   ; XY plane arcs
G64 P0.5 ; Blend move mode, 0.5 mm tolerance

(Movement and output commands)
G0 X60 Y110 F1200 ; Rapid move at 1200 mm/minute
M3 $1 ; Start tool 1 in clockwise direction
G1 X110 Y110 F1000 ; Travel move at 1000 mm/minute
G2 X110 Y10 I100 J60 ; Clockwise arc
G1 X60 Y10
G2 X60 Y110 I60 J60
G4 P1.0 ; Dwell for 1 second
M5 $1 ; Stop tool 1
G0 X1 Y1
"""

# Optional: If your path contains M3-4-5 commands,
# define device,port and on value of both tool directions.
# clockwise must be defined for M3 commands
clockwise = DIGITAL_OUTPUT_STATE(deviceId=1, port=0, value=1)
counterClockwise = DIGITAL_OUTPUT_STATE(1, 1, 1)
tool1 = TOOL(1, clockwise, counterClockwise)
# associate with the parent drive
driveAssociation = {"X": 1, "Y": 3}
# Configure the path following mode
mm.configPathMode(driveAssociation, tool1)
maxPathAcceleration = 1000  # mm/s^2

mm.setAxisMaxAcceleration(1,1000)
mm.setAxisMaxAcceleration(3,1000)
mm.setAxisMaxSpeed(1,500)
mm.setAxisMaxSpeed(3,500)

mm.moveToHome(1)
mm.moveToHome(3)
mm.waitForMotionCompletion()

# Start Path Following.
mm.startPath(GCODE_STRING, maxPathAcceleration)
running = True
while running:
    pathStatus = mm.getPathStatus()
    running = pathStatus['running']
    print('--> Got Path Status: ', pathStatus)
    time.sleep(0.5)

mm.stopPath()
```

```
print("--> Done!")
```

class ActuatorConfigParams

__init__(self, axisType, motorSize, brake, tuningProfile, gearbox, motorCurrent, gain, controlLoop, homingSpeed)

- **desc** Creates a class can be used to configure drives on a Machine Motion.
  - **params**
    - **axisType**
      - **desc** Type of the axis to be configured (impacts the default configuration values)
      - **type** String of the AXIS_TYPE class
    - **motorSize**
      - **desc** Size of the motor connected to the drive
      - **type** String of the MOTOR_SIZE class
    - **brake**
      - **desc** The brake installed on the connected actuator
      - **type** String of the Brake class None
    - **tuningProfile**
      - **desc** The tuning profile to be assigned to the motor connected to the drive
      - **type** String of the TUNING_PROFILES class None
    - **gearbox**
      - **desc** The gearbox ratio of the gearbox connected to the motor connected to the drive
      - **type** Number of the GEARBOX class None
    - **motorCurrent**
      - **desc** The motorCurrent to be assigned to the motor connected to the drive
      - **type** number None
    - **gain**

- **desc** The gain to be assigned to the motor connected to the drive
- **note** Only for actuators with type CUSTOM
-

  type number | None

- **controlLoop**
  - **desc** The control loop used by the motor connected to the drive
  - **note** Only for actuators with type CUSTOM
  -

    type String of the CONTROL_LOOPS class | None

- **homingSpeed**
  - **desc** The speed of the homing motion of the actuator in mm/s
  -

    type number | None

- **compatibility** MachineMotion v2.

Example: configActuator.py

```
import sys
sys.path.append("../")
from MachineMotion import *

### This Python example configures actuators for a MachineMotion v2. ###
mm = MachineMotionV2()

# Configure a roller conveyor with a single drive with minimum parameters

# The following configuration is added
# Axis 1:
#   Drive Number(s): 1
#   Axis Type: Roller Conveyor
#   Motor Size: Large Servo
print("--> Configuring single drive roller conveyor.")
singleDefaultConfig = ActuatorConfigParams(
    AXIS_TYPE.ROLLER_CONVEYOR,
    MOTOR_SIZE.LARGE
)
singleDefaultDrive = DriveConfigParams(
    AXIS_NUMBER.DRIVE1,
    DIRECTION.NORMAL
)
mm.configActuator(singleDefaultConfig, singleDefaultDrive)

# Configure a CUSTOM actuator.

# The following configuration is added
# Axis 2:
#   Drive Number(s): 2
#   Axis Type: Custom
#   Motor Size: Medium Servo
#   Brake: None
```

```
#   Brake: None
#   Tuning Profile: Default
#   Motor Current: 10 A
#   Gain: 100 mm/turn
#   Control Loop: Closed Loop
print("--> Configuring custom axis.")
motorCurrent = 10  # A
mechGain = 100  # mm/turn

customConfig = ActuatorConfigParams(
    axisType=AXIS_TYPE.CUSTOM,
    motorSize=MOTOR_SIZE.MEDIUM,
    brake=BRAKE.NONE,
    tuningProfile=TUNING_PROFILES.DEFAULT,
    motorCurrent=motorCurrent,
    gain=mechGain,
    controlLoop=CONTROL_LOOPS.CLOSED_LOOP,
)
customDrive = DriveConfigParams(
    AXIS_NUMBER.DRIVE2,
    DIRECTION.REVERSE
)
mm.configActuator(customConfig, customDrive)

# Configure a multi-drive timing belt.
# The following configuration is added
# Axis 3:
#   Drive Number(s): 3, 4
#   Axis Type: Timing Belt
#   Motor Size: Small Servo
#   Gearbox: 1/5 Gearbox
print("--> Configuring multidrive timing belt.")
multidriveConfig = ActuatorConfigParams(
    axisType=AXIS_TYPE.TIMING_BELT,
    motorSize=MOTOR_SIZE.SMALL,
    gearbox=GEARBOX.RATIO_5_TO_1,
)
multidriveParentDrive = DriveConfigParams(AXIS_NUMBER.DRIVE3, DIRECTION.NORMAL)
multidriveChildDrive = DriveConfigParams(AXIS_NUMBER.DRIVE4, DIRECTION.REVERSE)
mm.configActuator(multidriveConfig, multidriveParentDrive, multidriveChildDrive)

print("--> Configuration complete!")
```

class DriveConfigParams

__init__(self, driveNumber, direction)

- **desc** Creates an object representing the configuration for a specific drive on a Machine Motion.
  - **params**
    - **driveNumber**
      - **desc** The number of the drive to be configured
      - **type** Number of the AXIS_NUMBER class
    - **direction**
      - **desc** Defines the direction of motion of the drive
      - **type** String of the DIRECTION class

- **compatibility** MachineMotion v2.

Example: configActuator.py

```
import sys
sys.path.append("../")
from MachineMotion import *

### This Python example configures actuators for a MachineMotion v2. ###
mm = MachineMotionV2()

# Configure a roller conveyor with a single drive with minimum parameters

# The following configuration is added
# Axis 1:
#   Drive Number(s): 1
#   Axis Type: Roller Conveyor
#   Motor Size: Large Servo
print("--> Configuring single drive roller conveyor.")
singleDefaultConfig = ActuatorConfigParams(
    AXIS_TYPE.ROLLER_CONVEYOR,
    MOTOR_SIZE.LARGE
)
singleDefaultDrive = DriveConfigParams(
    AXIS_NUMBER.DRIVE1,
    DIRECTION.NORMAL
)
mm.configActuator(singleDefaultConfig, singleDefaultDrive)

# Configure a CUSTOM actuator.

# The following configuration is added
# Axis 2:
#   Drive Number(s): 2
#   Axis Type: Custom
#   Motor Size: Medium Servo
#   Brake: None
#   Tuning Profile: Default
#   Motor Current: 10 A
#   Gain: 100 mm/turn
#   Control Loop: Closed Loop
print("--> Configuring custom axis.")
motorCurrent = 10  # A
mechGain = 100  # mm/turn

customConfig = ActuatorConfigParams(
    axisType=AXIS_TYPE.CUSTOM,
    motorSize=MOTOR_SIZE.MEDIUM,
    brake=BRAKE.NONE,
    tuningProfile=TUNING_PROFILES.DEFAULT,
    motorCurrent=motorCurrent,
    gain=mechGain,
    controlLoop=CONTROL_LOOPS.CLOSED_LOOP,
)
customDrive = DriveConfigParams(
    AXIS_NUMBER.DRIVE2,
    DIRECTION.REVERSE
)
mm.configActuator(customConfig, customDrive)

# Configure a multi-drive timing belt.
# The following configuration is added
# Axis 3:
#   Drive Number(s): 3, 4
#   Axis Type: Timing Belt
#   Motor Size: Small Servo
```

```
#   Gearbox: 1/5 Gearbox
print("--> Configuring multidrive timing belt.")
multidriveConfig = ActuatorConfigParams(
    axisType=AXIS_TYPE.TIMING_BELT,
    motorSize=MOTOR_SIZE.SMALL,
    gearbox=GEARBOX.RATIO_5_TO_1,
)
multidriveParentDrive = DriveConfigParams(AXIS_NUMBER.DRIVE3, DIRECTION.NORMAL)
multidriveChildDrive = DriveConfigParams(AXIS_NUMBER.DRIVE4, DIRECTION.REVERSE)
mm.configActuator(multidriveConfig, multidriveParentDrive, multidriveChildDrive)

print("--> Configuration complete!")
```

class CONTROL_DEVICE_SIGNALS

Variables

```
SIGNAL0 = "SIGNAL0"
SIGNAL1 = "SIGNAL1"
SIGNAL2 = "SIGNAL2"
SIGNAL3 = "SIGNAL3"
SIGNAL4 = "SIGNAL4"
SIGNAL5 = "SIGNAL5"
SIGNAL6 = "SIGNAL6"
```

class CONTROL_DEVICE_TYPE

Variables

```
IO_EXPANDER_GENERIC = "IO_EXPANDER_GENERIC"
ENCODER = "ENCODER"
```

class CONTROL_DEVICE_PORTS

Variables

```
SENSOR4 = "SENSOR4"
SENSOR5 = "SENSOR5"
SENSOR6 = "SENSOR6"
```

class NETWORK_MODE

Variables

```
static = "static"
dhcp = "dhcp"
```

Examples

moveRelative.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases relative moves with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Begin Relative Move
distance = 100  # in mm
mm.moveRelative(axis, distance)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " moved " + str(distance) + "mm")

# Pass a negative distance value to move in the opposite direction
distance = -100  # in mm
mm.moveRelative(axis, distance)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " moved " + str(distance) + "mm")

print("--> Example Complete")
```

oneDriveControl.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to control a motor with a One Drive MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

### MachineMotion configuration ###
mm = MachineMotionV2OneDrive()

axis = AXIS_NUMBER.DRIVE1
motorCurrent = 10  # current (A)

### Home axis 1 specifically
print("--> Homing axis " + str(axis))
mm.moveToHome(axis)
mm.waitForMotionCompletion()

### Relative move axis 1
print("--> Moving axis " + str(axis) + " by 100mm.")
mm.moveRelative(axis, 100)
mm.waitForMotionCompletion()

### Absolute move axis 1
position = 50  # in mm
mm.moveToPosition(axis, position)
print("--> Axis " + str(axis) + " is moving towards position " + str(position) + "mm")
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " is at position " + str(position) + "mm")

### Getting position for axis 1
print("--> Read the position of axis " + str(axis))
actualPosition = mm.getActualPositions(axis)
print("--> Actual position is: " + str(actualPosition))

### Controlling any other axis will yield an error:
try:
    print("--> Controlling an axis that doesn't exist..")
    mm.moveToPosition(2, position)
except Exception as e:
    print(e)

print("Example Complete!")
```

moveContinuous.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases continuous moves with MachineMotion v2. ###
```

```
# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Heavy Duty Roller Conveyor
#   Motor Size: Large Servo


# Create MachineMotion instance
mm = MachineMotionV2()

conveyor_axis = AXIS_NUMBER.DRIVE1

# Note: on MachineMotion software version 2.4.0 and newer, continuous moves are limited by the axis max speed and acceleration.
# Please see the section below for more details.

### CONTINUOUS MOVES ###
# Start the continuous move
print("Start Conveyor Move...")
print("Continuous move: speed 100mm/s & acceleration 100mm/s^2")
mm.moveContinuous(conveyor_axis, 100, 100)
time.sleep(5)

# Change speed while moving
print("Continuous move: speed 500mm/s & acceleration 250mm/s^2")
mm.moveContinuous(conveyor_axis, 500, 250)
time.sleep(5)

# Reverse direction of conveyor by changing the sign of the speed
print("Reverse continuous move: speed -1000mm/s & acceleration 500mm/s^2")
mm.moveContinuous(conveyor_axis, -1000, 500)
time.sleep(5)

# Or pass in the optional direction argument
print("Reverse continuous move: speed -500mm/s & acceleration 500mm/s^2")
mm.moveContinuous(conveyor_axis, 500, 500, direction=DIRECTION.NEGATIVE)
time.sleep(5)

# Stop the continuous move
print("Stop Conveyor Move: acceleration 500mm/s^2")
mm.stopMoveContinuous(conveyor_axis, 500)
time.sleep(3)

# # For MachineMotion Software version >= 2.4.0 only:

# # Setting the axis max speed and acceleration will limit continuous moves:
# mm.setAxisMaxSpeed(conveyor_axis, 1000)
# mm.setAxisMaxAcceleration(conveyor_axis, 1000)

# # Start the continuous move at max speed and acceleration
# mm.moveContinuous(conveyor_axis)
# time.sleep(5)

# # Slow down conveyor
# mm.setAxisMaxSpeed(conveyor_axis, 500)
# time.sleep(5)

# # Reverse direction of conveyor
# mm.moveContinuous(conveyor_axis, direction=DIRECTION.NEGATIVE)
# time.sleep(5)

# # Stop the continuous move
# mm.stopMoveContinuous(conveyor_axis)
# time.sleep(5)

print("--> Example completed")
```

getPositions.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to read actuator positions with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 2, 3, 4
#   Any valid configuration

mm = MachineMotionV2()

# Configure actuator
axis = AXIS_NUMBER.DRIVE1

# Home Axis Before Moving
print("--> Axis " + str(axis) + " moving home")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("--> Axis " + str(axis) + " homed")


###### READ POSITIONS ######

# Read the position of one axis
print("--> Read the position of one axis")
actualPosition_axis = mm.getActualPositions(axis)
print(
    "Actual position of axis "
    + str(axis)
    + " is : "
    + str(actualPosition_axis)
    + " mm."
)

# Read the position of several axes
print("--> Read the position of several axes")
actualPositions = mm.getActualPositions()
print("Actual position of axis 1 is : " + str(actualPositions[1]) + " mm.")
print("Actual position of axis 2 is : " + str(actualPositions[2]) + " mm.")
print("Actual position of axis 3 is : " + str(actualPositions[3]) + " mm.")
print("Actual position of axis 4 is : " + str(actualPositions[4]) + " mm.")

###### MOVE AND READ POSITIONS ######

# Define Motion Parameters
distance = 100

# Move 100mm and check position again
mm.moveRelative(axis, distance)
print("--> Move ongoing")
while not mm.isMotionCompleted():
    actualPosition_axis = mm.getActualPositions(axis)
```

```
    actualPosition_axis = mm.getActualPositions(axis)
    print(
        "Actual position of axis "
        + str(axis)
        + " is : "
        + str(actualPosition_axis)
        + " mm."
    )

mm.waitForMotionCompletion()
print("--> Move completed")
actualPosition_axis = mm.getActualPositions(axis)
print(
    "Actual position of axis "
    + str(axis)
    + " is : "
    + str(actualPosition_axis)
    + " mm."
)

print("--> End of example")
```

getEndStopState.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to read enstop sensor states with MachineMotion v2. ###

mm = MachineMotionV2()

# Get End Stop State
endstopStates = mm.getEndStopState()
# If the direction of the axis is normal,
#   then the home sensor is the "_min" sensor, and the end sensor is the "_max" sensor.
# If the direction of the axis reversed,
#   then the home sensor is the "_max" sensor, and the end sensor is the "_min" sensor.
axis1_home_sensor_status = endstopStates["x_min"]
axis1_endstop_sensor_status = endstopStates["x_max"]
axis2_home_sensor_status = endstopStates["y_min"]
axis2_endstop_sensor_status = endstopStates["y_max"]
axis3_home_sensor_status = endstopStates["z_min"]
axis3_endstop_sensor_status = endstopStates["z_max"]
axis4_home_sensor_status = endstopStates["w_min"]
axis4_endstop_sensor_status = endstopStates["w_max"]

print("Axis 1 : " + "Home sensor is : " + str(axis1_home_sensor_status))
print("Axis 1 : " + "End sensor is : " + str(axis1_endstop_sensor_status))
print("Axis 2 : " + "Home sensor is : " + str(axis2_home_sensor_status))
print("Axis 2 : " + "End sensor is : " + str(axis2_endstop_sensor_status))
print("Axis 3 : " + "Home sensor is : " + str(axis3_home_sensor_status))
print("Axis 3 : " + "End sensor is : " + str(axis3_endstop_sensor_status))
print("Axis 4 : " + "Home sensor is : " + str(axis4_home_sensor_status))
print("Axis 4 : " + "End sensor is : " + str(axis4_endstop_sensor_status))
```

stopAllMotion.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to stop motion on MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Begin Relative Move
distance = 1000
mm.moveRelative(axis, distance)
print("Axis " + str(axis) + " is moving " + str(distance) + "mm")
# This move should take 5 seconds to complete (distance/speed). Instead, we wait 2 seconds and then stop the machine.
time.sleep(2)
mm.stopAllMotion()
print("Axis " + str(axis) + " stopped.")
```

moveIndependentAxis.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### This Python example configures and moves an independent axis.
### The individual axis speed and acceleration are set,
### then motion on the axis is selectively started, stopped and monitored

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large Servo

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Move, stop and monitor the axis
axisSpeed = 50  # mm/s
axisAcceleration = 50  # mm/s^2
distance = 250  # mm

print(
    "Configuring axis: ",
    axis,
    " speed: ",
    axisSpeed,
    "mm/s",
    " acceleration: ",
    axisAcceleration,
    "mm/s^2",
)
mm.setAxisMaxSpeed(axis, axisSpeed)
mm.setAxisMaxAcceleration(axis, axisAcceleration)

print("Relative move: ", distance, "mm")
mm.moveRelative(axis, distance)

axisMotionCompleted = mm.isMotionCompleted(axis)
if axisMotionCompleted:
    print("Axis ", axis, " is NOT currently moving.")
else:
    print("Axis ", axis, " is currently moving.")

time.sleep(3)

print("Stopping just axis: ", axis)
mm.stopAxis(axis)

time.sleep(3)

print("Moving and waiting for axis: ", axis)
mm.moveToPosition(axis, distance)
mm.waitForMotionCompletion(axis)

print("Done!")
```

moveToHomeAll.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to home actuators with MachineMotion v2. ###

### MachineMotion configuration ###

mm = MachineMotionV2()

### Home all axes simultaneously
print("All axes moving home simultaneously")
mm.moveToHomeAll()
mm.waitForMotionCompletion()
print("All axes homed")
```

moveToHome.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to home actuators with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Home the actuator
print("Axis " + str(axis) + " is going home")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " is at home")
```

speedAndAcceleration.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### IMPORTANT: This example file is compatible with MachineMotion Software version 2.4.0 or newer. ###

### speedAndAcceleration.py
# This example shows how to change the configured max speed and max acceleration for an axis,
# as well as how to retrieve the configured max speed, max acceleration and actual speed of a specific axis.

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

maxSpeedAxis = 50  # mm/s
maxAccelerationAxis = 150  # mm/s^2

# Setting speeds and accelerations

print("Setting max speed for axis ", axis, ": ", maxSpeedAxis, "mm/s")
mm.setAxisMaxSpeed(axis, maxSpeedAxis)
print("Setting max acceleration for axis ", axis, ": ", maxAccelerationAxis, "mm/s^2")
mm.setAxisMaxAcceleration(axis, maxAccelerationAxis)

# Getting speeds and accelerations

print("Getting speeds and accelerations...")

maxSpeed = mm.getAxisMaxSpeed(axis)
maxAccel = mm.getAxisMaxAcceleration(axis)
print(
    "For axis ",
    axis,
    " retrieved max speed: ",
    maxSpeed,
    "mm/s, max acceleration: ",
    maxAccel,
    "mm/s^2",
)

actualSpeed = mm.getActualSpeeds(axis)
print(
    "For axis ", axis, " retrieved actual speed: ", actualSpeed, "mm/s"
)  # The value should be 0 as the axis is not moving.

print("Getting actual speeds for all axes, in mm/s:")
actualSpeeds = mm.getActualSpeeds()
print(actualSpeeds)  # The values should be 0 as the axes are not moving.

print("Done!")
```

moveToPosition.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases moveToPosition with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1  # The axis that you'd like to move

# Movement configuration
position = 100  # The absolute position to which you'd like to move

# Home Axis before move to position
print("Axis " + str(axis) + " is going home.")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " homed.")

# Move
mm.moveToPosition(axis, position)
print("Axis " + str(axis) + " is moving towards position " + str(position) + "mm")
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " is at position " + str(position) + "mm")
```

moveToPositionCombined.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases combined absolute moves with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 2
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 3
#   Drive Number(s): 3
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axesToMove = [AXIS_NUMBER.DRIVE1, AXIS_NUMBER.DRIVE2, AXIS_NUMBER.DRIVE3]

# Home actuators before performing absolute moves
print("Axes Moving Home Sequentially")
for axis in axesToMove:
    mm.moveToHome(axis)
    mm.waitForMotionCompletion()
print("Axes homed")

# Simultaneously moves three axis:
#   Moves axis 1 to absolute position 50mm
#   Moves axis 2 to absolute position 100mm
#   Moves axis 3 to absolute position 50mm
positions = [50, 100, 50]
mm.moveToPositionCombined(axesToMove, positions)
mm.waitForMotionCompletion()
for index, axis in enumerate(axesToMove):
    print("Axis " + str(axis) + " moved to position " + str(positions[index]) + "mm")
```

moveRelativeCombined.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases combined relative moves with MachineMotion v1. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 2
#   Axis Type: Timing Belt
#   Motor Size: Large
# Axis 3
#   Drive Number(s): 3
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axesToMove = [AXIS_NUMBER.DRIVE1, AXIS_NUMBER.DRIVE2, AXIS_NUMBER.DRIVE3]

# Simultaneously moves three axis:
#   Move axis 1 in the positive direction by 50 mm
#   Move axis 2 in the negative direction by 100 mm
#   Move axis 3 in the positive direction by 50 mm
distances = [50, 100, 50]
mm.moveRelativeCombined(axesToMove, distances)
mm.waitForMotionCompletion()
for index, axis in enumerate(axesToMove):
    print("Axis " + str(axis) + " moved " + str(distances[index]) + "mm")
```

setPosition.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to set actuator position with MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Home the actuator
print("Axis " + str(axis) + " will home")
mm.moveToHome(axis)
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " homed")

### Perform asolute moves with different reference points ###

print("Absolute Moves are referenced from home")
position = 100
mm.moveToPosition(axis, position)
mm.waitForMotionCompletion()
print("Axis " + str(axis) + " is " + str(position) + "mm away from home.")

# Change the reference point to the current position
mm.setPosition(axis, 0)
print(
    "Absolute moves on axis "
    + str(axis)
    + " are now referenced from "
    + str(position)
    + "mm from home. "
)
time.sleep(2)

# Move again
position2 = 30
print(
    "Now moving to absolute position "
    + str(position2)
    + " mm, referenced from location 'setPosition' was called"
)
mm.moveToPosition(axis, position2)
mm.waitForMotionCompletion()
print(
    "Axis "
    + str(axis)
    + " is now "
    + str(position2)
    + "mm from reference position and "
    + str(position + position2)
    + "mm from home"
)
```

waitForMotionCompletion.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how to wait for actuator motion completion on MachineMotion v2. ###

# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1
#   Axis Type: Timing Belt
#   Motor Size: Large

mm = MachineMotionV2()

axis = AXIS_NUMBER.DRIVE1

# Move the axis by 100mm
distance = 100

print("Moving %d mm!" % distance)
mm.moveRelative(axis, distance)
print("This message gets printed immediately")
mm.waitForMotionCompletion()
print("This message gets printed once machine has finished moving")
```

configActuator.py

```
import sys
sys.path.append("../")
from MachineMotion import *

### This Python example configures actuators for a MachineMotion v2. ###
mm = MachineMotionV2()

# Configure a roller conveyor with a single drive with minimum parameters

# The following configuration is added
# Axis 1:
#   Drive Number(s): 1
#   Axis Type: Roller Conveyor
#   Motor Size: Large Servo
print("--> Configuring single drive roller conveyor.")
singleDefaultConfig = ActuatorConfigParams(
    AXIS_TYPE.ROLLER_CONVEYOR,
    MOTOR_SIZE.LARGE
)
singleDefaultDrive = DriveConfigParams(
    AXIS_NUMBER.DRIVE1,
    DIRECTION.NORMAL
)
mm.configActuator(singleDefaultConfig, singleDefaultDrive)

# Configure a CUSTOM actuator.

# The following configuration is added
```

```
# Axis 2:
#   Drive Number(s): 2
#   Axis Type: Custom
#   Motor Size: Medium Servo
#   Brake: None
#   Tuning Profile: Default
#   Motor Current: 10 A
#   Gain: 100 mm/turn
#   Control Loop: Closed Loop
print("--> Configuring custom axis.")
motorCurrent = 10  # A
mechGain = 100  # mm/turn

customConfig = ActuatorConfigParams(
    axisType=AXIS_TYPE.CUSTOM,
    motorSize=MOTOR_SIZE.MEDIUM,
    brake=BRAKE.NONE,
    tuningProfile=TUNING_PROFILES.DEFAULT,
    motorCurrent=motorCurrent,
    gain=mechGain,
    controlLoop=CONTROL_LOOPS.CLOSED_LOOP,
)
customDrive = DriveConfigParams(
    AXIS_NUMBER.DRIVE2,
    DIRECTION.REVERSE
)
mm.configActuator(customConfig, customDrive)

# Configure a multi-drive timing belt.
# The following configuration is added
# Axis 3:
#   Drive Number(s): 3, 4
#   Axis Type: Timing Belt
#   Motor Size: Small Servo
#   Gearbox: 1/5 Gearbox
print("--> Configuring multidrive timing belt.")
multidriveConfig = ActuatorConfigParams(
    axisType=AXIS_TYPE.TIMING_BELT,
    motorSize=MOTOR_SIZE.SMALL,
    gearbox=GEARBOX.RATIO_5_TO_1,
)
multidriveParentDrive = DriveConfigParams(AXIS_NUMBER.DRIVE3, DIRECTION.NORMAL)
multidriveChildDrive = DriveConfigParams(AXIS_NUMBER.DRIVE4, DIRECTION.REVERSE)
mm.configActuator(multidriveConfig, multidriveParentDrive, multidriveChildDrive)

print("--> Configuration complete!")
```

digitalRead.py

```
import sys

sys.path.append("../")
from MachineMotion import *

### This Python example reads digital inputs for MachineMotion v2. ###

mm = MachineMotionV2()

# Detect all connected digital IO Modules
detectedIOModules = mm.detectIOModules()

# Read and print all input values
if detectedIOModules is not None:
    for IO_Name, IO_NetworkID in detectedIOModules.items():
        readPins = [0, 1, 2, 3]
        for readPin in readPins:
            pinValue = mm.digitalRead(IO_NetworkID, readPin)
            print(
                "Pin " + str(readPin) + " on " + IO_Name + " has value " + str(pinValue)
            )
```

digitalWrite.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example writes digital outputs for MachineMotion v2. ###

mm = MachineMotionV2()

# Detect all connected digital IO Modules
detectedIOModules = mm.detectIOModules()

# Toggles the output pins
if detectedIOModules is not None:
    for IO_Name, IO_NetworkID in detectedIOModules.items():
        writePins = [0, 1, 2, 3]
        for writePin in writePins:
            print(
                "Pin " + str(writePin) + " on " + IO_Name + " is going to flash twice"
            )

            mm.digitalWrite(IO_NetworkID, writePin, 1)
            time.sleep(1)
            mm.digitalWrite(IO_NetworkID, writePin, 0)
            time.sleep(1)
            mm.digitalWrite(IO_NetworkID, writePin, 1)
            time.sleep(1)
            mm.digitalWrite(IO_NetworkID, writePin, 0)
            time.sleep(1)
```

powerSwitch.py

```
import sys
import time

sys.path.append("../")
from MachineMotion import *

### This Python example control a power switch module on MachineMotion v2. ###

mm = MachineMotionV2()

deviceNetworkId = 7

### Set the power switch
print("--> Turning power switch on")
mm.setPowerSwitch(deviceNetworkId, POWER_SWITCH.ON)
time.sleep(3)
print("--> Turning power switch off")
mm.setPowerSwitch(deviceNetworkId, POWER_SWITCH.OFF)
time.sleep(3)

print("--> Example Complete")
```

pushButton.py

```
import sys
import time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases different uses for a push button module on MachineMotion v2. ###
mm = MachineMotionV2()

### Define the module and button you would like to monitor.
deviceNetworkId = 5
blackButton = PUSH_BUTTON.COLOR.BLACK
whiteButton = PUSH_BUTTON.COLOR.WHITE

### Block the script until a button is pushed:
print("--> Waiting 5 seconds for the white button to be pushed!")
buttonWasPushed = mm.waitOnPushButton(
    deviceNetworkId, whiteButton, PUSH_BUTTON.STATE.PUSHED, 5
)
if buttonWasPushed:
    print("--> White button was pushed!")
else:
    print("--> waitOnPushButton timed out!")

time.sleep(1)

### Manually read the state of a push button
buttonState = mm.readPushButton(deviceNetworkId, whiteButton)
print("--> Reading from white push button: " + buttonState)
time.sleep(2)

### Define a callback to process push button status
def templateCallback(button_state):
    print(
        "--> templateCallback: Black push button on module: "
        + str(deviceNetworkId)
        + " has changed state."
    )
    if button_state == PUSH_BUTTON.STATE.PUSHED:
        print("--> Black push button has been pushed.")
    elif button_state == PUSH_BUTTON.STATE.RELEASED:
        print("--> Black push button has been released.")


###  You must first bind the callback function!
mm.bindPushButtonEvent(deviceNetworkId, blackButton, templateCallback)

print("--> Push the black button to trigger event! Press ctrl-c to exit")

while True:
    time.sleep(0.5)
```

eStop.py

```
#!/usr/bin/python
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example showcases how monitor eStop on MachineMotion v2. ###

# Create MachineMotion instance
mm = MachineMotionV2()
time.sleep(0.1)  # Wait to initialize internal eStop topics.

# Define a callback to process estop status
def handleEstopStatus(status):
    if status == True:
        # executed when you enter estop
        print("MachineMotion is in estop!")
        print("Estop must be reset with a physical module")
    if status == False:
        # executed when you exit estop
        print("MachineMotion not in estop...")
        print("Estop must be triggered with a physical module")


mm.bindeStopEvent(handleEstopStatus)

while True:
    time.sleep(1)
```

controlBrakes.py

```
import sys, time

sys.path.append("../")
from MachineMotion import *

### This Python example control brakes on MachineMotion v2. ###

mm = MachineMotionV2()

# Define the brake parameters
axis = 1  # Drive number
safety_adapter_presence = True  # Is a yellow safety adapter plugged in between the brake cable and the brake port
# Important Note : This flag must always be set to "True" on MachineMotions v2.

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)

# Lock the brake
mm.lockBrake(axis, safety_adapter_presence)
time.sleep(0.2)
# Wait before reading brake state, for it to get correctly updated

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)

# DO NOT MOVE WHILE BRAKE IS ENGAGED
print("Waiting two seconds. Do not move the actuator while the brake is locked !")
time.sleep(2)  # Unlock the brake after two seconds

# Release the brakes
mm.unlockBrake(axis, safety_adapter_presence)
time.sleep(0.2)
# Wait before reading brake state, for it to get correctly updated

# Read brake state
brake_state = mm.getBrakeState(axis, safety_adapter_presence)
print("The brake connected to port " + str(axis) + " is : " + brake_state)
```

pathFollowing.py

```
import sys
sys.path.append("../")
from MachineMotion import *

mm = MachineMotionV2()

### This example demonstrates the creation and execution of a simple 2D path ###
# Expected configuration for this example (via Control Center)
# Axis 1
#   Drive Number(s): 1, 2
#   Axis Type: Enclosed Timing Belt with 5:1 gearbox
#   Motor Size: Large
# Axis 2
#   Drive Number(s): 3
#   Axis Type: Enclosed Ballscrew
#   Motor Size: Large
```

```
GCODE_STRING = """
(Operational mode commands)
G90   ; Absolute position mode
G90.1 ; Absolute arc centre
G21   ; Use millimeter units
G17   ; XY plane arcs
G64 P0.5 ; Blend move mode, 0.5 mm tolerance

(Movement and output commands)
G0 X60 Y110 F1200 ; Rapid move at 1200 mm/minute
M3 $1 ; Start tool 1 in clockwise direction
G1 X110 Y110 F1000 ; Travel move at 1000 mm/minute
G2 X110 Y10 I100 J60 ; Clockwise arc
G1 X60 Y10
G2 X60 Y110 I60 J60
G4 P1.0 ; Dwell for 1 second
M5 $1 ; Stop tool 1
G0 X1 Y1
"""

# Optional: If your path contains M3-4-5 commands,
# define device,port and on value of both tool directions.
# clockwise must be defined for M3 commands
clockwise = DIGITAL_OUTPUT_STATE(deviceId=1, port=0, value=1)
counterClockwise = DIGITAL_OUTPUT_STATE(1, 1, 1)
tool1 = TOOL(1, clockwise, counterClockwise)
# associate with the parent drive
driveAssociation = {"X": 1, "Y": 3}
# Configure the path following mode
mm.configPathMode(driveAssociation, tool1)
maxPathAcceleration = 1000  # mm/s^2

mm.setAxisMaxAcceleration(1,1000)
mm.setAxisMaxAcceleration(3,1000)
mm.setAxisMaxSpeed(1,500)
mm.setAxisMaxSpeed(3,500)

mm.moveToHome(1)
mm.moveToHome(3)
mm.waitForMotionCompletion()

# Start Path Following.
mm.startPath(GCODE_STRING, maxPathAcceleration)
running = True
while running:
    pathStatus = mm.getPathStatus()
    running = pathStatus['running']
    print('--> Got Path Status: ', pathStatus)
    time.sleep(0.5)

mm.stopPath()
print("--> Done!")
```