Application Programming Interface: Python v4.3

Contents

Application Programming Interface: Python v4.3

Machine Motion V2

Overview

This is the developers reference manual for version 4.3 of MachineMotion's Python API. This API is compatible with Python version 3.6 and later.
If this is your first time using the MachineMotion controller, please follow the Python programming manual here.
Version 4.3 of the API is compatible with MachineMotion controller software version 1.12 and later. 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 4.3 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
DEFAULT_TIMEOUT = 65
```

class MachineMotion(object)

back to API Reference

moveContinuous(axis, speed, accel)

Starts an axis using speed mode.

- **params**
  - **axis**
    - **desc** Axis to move
    - **type** Number
  - **speed**
    - **desc** Speed to move the axis at in mm / sec
    - **type** Number
  - **accel**
    - **desc** Acceleration used to reach the desired speed, in mm / sec^2
    - **type** Number
- **compatibility** MachineMotion v1 and MachineMotion v2, software version 2.3.0 and newer.
- **examples**
  - moveContinuous.py
  - (V2) moveContinuous.py

back to class MachineMotion(object)

stopMoveContinuous(axis, accel)

Stops an axis using speed mode.

- **params**
  - **axis**
    - **desc** Axis to move
    - **type** Number
  - **accel**
    - **desc** Acceleration used to reach a null speed, in mm / sec^2
    - **type** Number
- **compatibility** MachineMotion v1 and MachineMotion v2.
- **examples**
  - moveContinuous.py

- (V2) moveContinuous.py

back to class MachineMotion(object)

getDesiredPositions(axis)

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.
- **examples**
  - getPositions.py
  - (V2) getPositions.py

back to class MachineMotion(object)

getActualPositions(axis)

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.
- **returnValueType** Number or Dictionary of numbers
- **compatibility** MachineMotion v1 and MachineMotion v2.
- **examples**
  - getPositions.py
  - (V2) getPositions.py

back to class MachineMotion(object)

getEndStopState()

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.
- **examples**
  - getEndStopState.py
  - (V2) getEndStopState.py

back to class MachineMotion(object)

stopAllMotion()

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.
- **examples**
  - stopAllMotion.py
  - (V2) stopAllMotion.py

back to class MachineMotion(object)

moveToHomeAll()

Initiates the homing sequence of all axes. All axes will move home sequentially.

- **compatibility** MachineMotion v1 and MachineMotion v2.
- **examples**
  - moveToHomeAll.py
  - (V2) moveToHomeAll.py

back to class MachineMotion(object)

moveToHome(axis)

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.
- **examples**
  - moveToHome.py
  - (V2) moveToHome.py

back to class MachineMotion(object)

setSpeed(speed, units)

Sets the global speed for all movement commands on all axes.

- **params**
  - **speed**
    - **desc** The global max speed in mm/sec, 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.
- **examples**
    - setSpeed.py
    - (V2) setSpeed.py

back to class MachineMotion(object)

setAcceleration(acceleration, units)

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.
- **examples**
    - setAcceleration.py
    - (V2) setAcceleration.py

back to class MachineMotion(object)

moveToPosition(axis, position)

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.
- **examples**
    - moveToPosition.py
    - (V2) moveToPosition.py

back to class MachineMotion(object)

moveToPositionCombined(axes, positions)

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.
- **examples**
    - moveToPositionCombined.py
    - (V2) moveToPositionCombined.py
- **note** The current speed and acceleration settings are applied to the combined motion of the axes.

back to class MachineMotion(object)

moveRelative(axis, distance)

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.
- **examples**
    - moveRelative.py
    - (V2) moveRelative.py

back to class MachineMotion(object)

moveRelativeCombined(axes, distances)

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.
- **examples**
    - moveRelativeCombined.py
    - (V2) moveRelativeCombined.py
- **note** The current speed and acceleration settings are applied to the combined motion of the axes.

back to class MachineMotion(object)

setPosition(axis, position)

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.
- **examples**
    - setPosition.py
    - (V2) setPosition.py

back to class MachineMotion(object)

emitgCode(gCode)

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.
- **examples**
    - emitgCode.py

back to class MachineMotion(object)

isMotionCompleted()

Indicates if the last move command has completed.

- **returnValue** Returns false if the machine is currently executing a movement command.
- **returnValueType** Boolean
- **compatibility** MachineMotion v1 and MachineMotion v2.
- **examples**
    - getPositions.py
    - (V2) getPositions.py
- **note** isMotionCompleted does not account for on-going continuous moves.

back to class MachineMotion(object)

waitForMotionCompletion()

Pauses python program execution until machine has finished its current movement.

- **compatibility** MachineMotion v1 and MachineMotion v2.
- **examples**
    - waitForMotionCompletion.py
    - (V2) waitForMotionCompletion.py
- **note** waitForMotionCompletion does not account for on-going continuous moves.

back to class MachineMotion(object)

configHomingSpeed(axes, speeds, units)

Sets homing speed for all selected axes.

- **params**
  - **axes**
    - **desc** A list of the axes to configure. ex - [1,2,3]
    - **type** List of Numbers
  - **speeds**
    - **desc** A list of homing speeds to set for each axis. ex - [50, 50, 100]
    - **type** List of Numbers
  - **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.
- **examples**
  - configHomingSpeed.py
  - (V2) configHomingSpeed.py
- **note** Once set, the homing speed will apply to all programs, including MachineLogic applications.

back to class MachineMotion(object)

configAxis(axis, uStep, mechGain)

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.
- **examples**
  - configAxis.py

back to class MachineMotion(object)

configAxisDirection(axis, direction)

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.
- **examples**
  - configAxisDirection.py

back to class MachineMotion(object)

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

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.
- **examples**
  - (V2) configStepperServo.py

back to class MachineMotion(object)

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

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

- **note** Warning, changing the configuration can de-energize motors and thus cause unintended behaviour on vertical axes.
- **compatibility** MachineMotion v2 only.
- **examples**
  - (V2) configStepperServo.py
  - (V2) configMultiDriveServo.py

back to class MachineMotion(object)

detectIOModules()

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.
- **examples**
  - digitalRead.py
  - (V2) digitalRead.py

back to class MachineMotion(object)

digitalRead(deviceNetworkId, pin)

Reads the state of a digital IO modules input pins.

- **params**
  - **deviceNetworkId**
    - **desc** The IO Modules device network ID. It can be found printed on the product sticker on the back of the digital IO module.
    - **type** Integer
  - **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.
- **examples**
  - digitalRead.py
  - (V2) digitalRead.py
- **note** 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,

digitalRead(deviceNetworkId, 2) will read the value on input pin 3.

back to class MachineMotion(object)

digitalWrite(deviceNetworkId, pin, value)

Sets voltage on specified pin of digital IO output pin to either logic HIGH (24V) or LOW (0V).

- **params**
  - **deviceNetworkId**
    - **desc** The IO Modules device network ID. It can be found printed on the product sticker on the back of the digital IO module.
    - **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.
- **examples**
  - digitalWrite.py
  - (V2) digitalWrite.py
- **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.

back to class MachineMotion(object)

setPowerSwitch(deviceNetworkId, switchState)

Sets a switch of a power switch module to ON (closed) or OFF (open).

- **params**
  - **deviceNetworkId**
    - **desc** Power switch device network ID. It can be found printed on the dipswitch of the power switch module.
    - **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.
- **examples**
  - (V2) powerSwitch.py

back to class MachineMotion(object)

waitOnPushButton(deviceNetworkId, button, state, timeout)

Wait until a push button has reached a desired state

- **params**
  - **deviceNetworkId**
    - **desc** The Push-Button module's device network ID. This is set on the dipswitch of the module.
    - **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.
- **examples**
    - (V2) pushButton.py

back to class MachineMotion(object)

bindPushButtonEvent(deviceNetworkId, button, callback_function)

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. This is set on the dipswitch of the module.
        - **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.
- **examples**
    - (V2) pushButton.py

back to class MachineMotion(object)

readPushButton(deviceNetworkId, button)

Reads the state of a Push-Button module button.

- **params**
    - **deviceNetworkId**
        - **desc** The Push-Button module's device network ID. This is set on the dipswitch of the module.
        - **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.
- **examples**
  - (V2) pushButton.py

back to class MachineMotion(object)

readEncoder(encoder, readingType)

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.
- **examples**
  - readEncoder.py
- **note** The encoder position returned by this function may be delayed by up to 250 ms due to internal propogation delays.

back to class MachineMotion(object)

triggerEstop()

Triggers the MachineMotion software emergency stop, cutting power to all drives and enabling brakes (if any). The software E stop must be released (using releaseEstop()) in order to re-enable the machine.

- **returnValue** The success of the operation.
- **returnValueType** Boolean
- **compatibility** MachineMotion v1 and MachineMotion v2.
- **examples**
  - eStop.py
  - (V2) eStop.py

back to class MachineMotion(object)

releaseEstop()

Releases the software E-stop and provides power back to the drives.

- **returnValue** The success of the operation.
- **returnValueType** Boolean
- **compatibility** MachineMotion v1 and MachineMotion v2.
- **examples**
  - eStop.py
  - (V2) eStop.py

back to class MachineMotion(object)

resetSystem()

Resets the system after an eStop event

- **returnValue** The success of the operation.
- **returnValueType** Boolean
- **compatibility** MachineMotion v1 and MachineMotion v2.
- **examples**
  - eStop.py
  - (V2) eStop.py

back to class MachineMotion(object)

bindeStopEvent(callback_function)

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.
- **examples**
  - eStop.py
  - (V2) eStop.py

back to class MachineMotion(object)

lockBrake(aux_port_number, safety_adapter_presence)

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.
- **examples**
  - controlBrakes.py
  - (V2) controlBrakes.py
- **note** This function is compatible only with V1F and more recent MachineMotions.

back to class MachineMotion(object)

unlockBrake(aux_port_number, safety_adapter_presence)

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.
- **examples**
  - controlBrakes.py
  - (V2) controlBrakes.py
- **note** This function is compatible only with V1F and more recent MachineMotions.

back to class MachineMotion(object)

getBrakeState(aux_port_number, safety_adapter_presence)

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.
- **examples**
  - controlBrakes.py
  - (V2) controlBrakes.py
- **note** This function is compatible only with V1F and more recent MachineMotions.

back to class MachineMotion(object)

class MACHINEMOTION_HW_VERSIONS

back to API Reference

Variables

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

class DIRECTION

back to API Reference

Variables

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

class AXIS_NUMBER

back to API Reference

Variables

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

class UNITS_SPEED

back to API Reference

Variables

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

class UNITS_ACCEL

back to API Reference

Variables

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

class DEFAULT_IP_ADDRESS

back to API Reference

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

back to API Reference

Variables

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

class MECH_GAIN

back to API Reference

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`

back to API Reference

Variables

```
steps_per_turn      = 200
```

class `AUX_PORTS`

back to API Reference

Variables

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

class `ENCODER_TYPE`

back to API Reference

Variables

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

class BRAKE_STATES

back to API Reference

Variables

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

class TUNING_PROFILES

back to API Reference

Variables

```
DEFAULT = "default"
CONVEYOR_TURNTABLE = "conveyor_turntable"
```

class CONTROL_LOOPS

back to API Reference

Variables

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

class POWER_SWITCH

back to API Reference

Variables

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

class PUSH_BUTTON

back to API Reference

class COLOR

back to class PUSH_BUTTON

Variables

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

class STATE

back to class PUSH_BUTTON

Variables

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

class MOTOR_SIZE

back to API Reference

Variables

```
SMALL   = "Small Servo"
MEDIUM  = "Medium Servo"
LARGE   = "Large Servo"
```

class MQTT

back to API Reference

Variables

```
TIMEOUT = 10.0
```

class PATH

back to class MQTT

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 CONTROL_DEVICE_SIGNALS

back to API Reference

Variables

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

class CONTROL_DEVICE_TYPE

back to API Reference

Variables

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

class CONTROL_DEVICE_PORTS

back to API Reference

Variables

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

class NETWORK_MODE

back to API Reference

Variables

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

Examples

configAxis.py

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

### This Python example configures an actuator for a MachineMotion v1. ###

mm = MachineMotion()

# Configure the axis number 1, 8 uSteps and 150 mm / turn for a timing belt
axis = AXIS_NUMBER.DRIVE1
uStep = MICRO_STEPS.ustep_8
mechGain = MECH_GAIN.timing_belt_150mm_turn
mm.configAxis(axis, uStep, mechGain)
print("Axis " + str(axis) + " configured with " + str(uStep) + " microstepping and " + str(mechGain) + "mm/turn mechanical gain")
```

configAxisDirection.py

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

### This Python example configures actuator direction for MachineMotion v1. ###

mm = MachineMotion()

#When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the axis number 1, 8 uSteps and 150 mm / turn for a timing belt
axis = AXIS_NUMBER.DRIVE1
uStep = MICRO_STEPS.ustep_8
mechGain = MECH_GAIN.timing_belt_150mm_turn
mm.configAxis(axis, uStep, mechGain)

homesTowards = {
    DIRECTION.NORMAL: "sensor " + str(axis) + "A",
    DIRECTION.REVERSE: "sensor " + str(axis) + "B"
}

# Change axis direction, and see how it affects subsequent moves
direction = DIRECTION.REVERSE
mm.configAxisDirection(axis, direction)
print("Axis " + str(axis) + " is set to " + direction + " mode. It will now home towards " + homesTowards[direction] + "." )
mm.moveToHome(axis)

direction = DIRECTION.NORMAL
mm.configAxisDirection(axis, direction)
print("Axis " + str(axis) + " is set to " + direction + " mode. It will now home towards " + homesTowards[direction] + "." )
mm.moveToHome(axis)
```

configHomingSpeed.py

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

### This Python example configures homing speed for MachineMotion v1. ###

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

### Configuring ###

axes = [1,2]
homingSpeeds = [50,100] # The homing speeds to set for each axis, in mm/sec

mm.configAxis(1, MICRO_STEPS.ustep_8, MECH_GAIN.timing_belt_150mm_turn)
mm.configAxis(2, MICRO_STEPS.ustep_8, MECH_GAIN.timing_belt_150mm_turn)
mm.configHomingSpeed(axes, homingSpeeds)    # Sets homing speeds for all selected axes.

### Testing the configuration ###

axis = 1    # The axis to move

print("Moving axis " + str(axis) + " by 100mm.")
mm.moveRelative(axis, 100)
mm.waitForMotionCompletion()

#Homes the axis at the newly configured homing speed.
print("Homing axis " + str(axis))
mm.moveToHome(axis)
```

controlBrakes.py

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

### This Python example control brakes on MachineMotion v1. ###

print(" ----- WARNING ------ Does your hardware version support brakes?")

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Define the brake parameters
brake_port_number = 1           # The AUX port number the brake is plugged in
safety_adapter_presence = True  # Is a yellow safety adapter plugged in between the brake cable and the AUX port

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

# Lock the brake
mm.lockBrake (brake_port_number, 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 (brake_port_number, safety_adapter_presence)
print ("The brake connected to AUX" + str(brake_port_number) + " 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 (brake_port_number, 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 (brake_port_number, safety_adapter_presence)
print ("The brake connected to AUX" + str(brake_port_number) + " is : " + brake_state)
```

digitalRead.py

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

### This Python example reads digital inputs for MachineMotion v1. ###

mm = MachineMotion()

# 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 v1. ###

mm = MachineMotion()

# 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)
```

eStop.py

```
#!/usr/bin/python
import sys, time
sys.path.append("../..")
from MachineMotion import *

### This Python example showcases how to control eStop on MachineMotion v1. ###

### MachineMotion CONFIGURATION ###

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

# Define a callback to process estop status
def templateCallback(estop_status):
    print("eStop status is : " + str(estop_status))
    if estop_status == True :
        # executed when you enter estop
        print("MachineMotion is in estop state.")
    elif estop_status == False :
        # executed when you exit estop
        print("MachineMotion is not in estop state.")
mm.bindeStopEvent(templateCallback)

### ESTOP TRIGGER ###

result = mm.triggerEstop()
if result == True   : print("--> Software stop triggered")
else                : print("--> Failed to trigger software stop")

time.sleep(3.0)

### ESTOP RELEASE ###

result = mm.releaseEstop()
if result == True   : print("--> Software stop released")
else                : print("--> Failed to release software stop")

time.sleep(3.0)

### SYSTEM RESET ###

result = mm.resetSystem()
if result == True   : print("--> System has been reset")
else                : print("--> Failed to reset system")

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

emitgCode.py

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

### This Python example showcases how to emit custom gcode with MachineMotion v1. ###

# Define a callback to process controller gCode responses (if desired)
def templateCallback(data):
   print ( "Controller gCode responses " + data )

mm = MachineMotion(DEFAULT_IP, gCodeCallback = templateCallback)

#When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Use the G2 to perform combined arc move on the first 2 axes
gCodeCommand = "G2 I10 J10"
reply = mm.emitgCode(gCodeCommand)
mm.waitForMotionCompletion()
print("G-code command '" + gCodeCommand + "' completed by MachineMotion.")
print("Reply is : " + reply)
```

getEndStopState.py

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

### This Python example showcases how to read enstop sensor states with MachineMotion v1. ###

mm = MachineMotion()

# When starting a program, one must remove the software stop
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# 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']

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) )
```

getPositions.py

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

### This Python example showcases how to read actuator positions with MachineMotion v1. ###

###### CONFIGURING MACHINEMOTION ######

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure actuator
axis = 1
print("--> Configuring actuator")
mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.rack_pinion_mm_turn)

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

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


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

# Read the position of one axis
print("--> Read the position of one axis")
desiredPosition_axis = mm.getDesiredPositions(axis)
actualPosition_axis  = mm.getActualPositions(axis)
print("Desired position of axis " + str(axis) + " is : " + str(desiredPosition_axis) + " mm.")
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")
desiredPositions = mm.getDesiredPositions()
actualPositions  = mm.getActualPositions()
print("Desired position of axis 1 is : " + str(desiredPositions[1]) + " mm.")
print("Desired position of axis 2 is : " + str(desiredPositions[2]) + " mm.")
print("Desired position of axis 3 is : " + str(desiredPositions[3]) + " mm.")
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.")


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

# Define Motion Parameters
distance = 500

# Move 500mm and check position again
mm.moveRelative(axis, distance)
desiredPosition_axis = mm.getDesiredPositions(axis)
print("--> Move ongoing")
print("Desired position of axis " + str(axis) + " is : " + str(desiredPosition_axis) + " mm.")
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")
desiredPosition_axis = mm.getDesiredPositions(axis)
actualPosition_axis  = mm.getActualPositions(axis)
print("Desired position of axis " + str(axis) + " is : " + str(desiredPosition_axis) + " mm.")
print("Actual position of axis " + str(axis) + " is : " + str(actualPosition_axis) + " mm.")

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

moveContinuous.py

```
#!/usr/bin/python
import sys, time
sys.path.append("../..")
from MachineMotion import *

### This Python example showcases continuous moves with MachineMotion v1. ###

### MachineMotion CONFIGURATION ###

# Create MachineMotion instance
mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure actuator
conveyor_axis = 1
mm.configAxis(conveyor_axis, MICRO_STEPS.ustep_8, MECH_GAIN.roller_conveyor_mm_turn)

### CONTINUOUS MOVES ###

conveyor_stop_acc = 500 # Deceleration of 500mm/s^2

print("Start Conveyor Move...")
print("Continuous move: speed 100mm/s & acceleration 50mm/s^2")
mm.moveContinuous(conveyor_axis, 100, 50)
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)

# Stop the continuous move
mm.stopMoveContinuous(conveyor_axis, conveyor_stop_acc)
time.sleep(2)

# 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)

# Stop the continuous move
print("Stop Conveyor Move...")
mm.stopMoveContinuous(conveyor_axis, conveyor_stop_acc)
time.sleep(3)
print("--> Example completed")
```

moveRelative.py

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

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

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = 1
mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.rack_pinion_mm_turn)

# 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.py

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

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

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure actuators
axesToMove = [1,2,3]
for axis in axesToMove:
    mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.timing_belt_150mm_turn)

# 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")
```

moveToHome.py

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

### This Python example showcases how to home actuators with MachineMotion v1. ###

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = AXIS_NUMBER.DRIVE1
mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.timing_belt_150mm_turn)

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

moveToHomeAll.py

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

### This Python example showcases how to home actuators with MachineMotion v1. ###

### MachineMotion configuration ###

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

### Home all axes sequentially
print ("All Axes Moving Home sequentially")
mm.moveToHomeAll()
print("All Axes Homed")
```

moveToPosition.py

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

### This Python example showcases moveToPosition with MachineMotion v1. ###

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Axis configuration
axis = 1            #The axis that you'd like to move
mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.timing_belt_150mm_turn)

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

# Home Axis before moving to position
print("Axis " + str(axis) + " is going to home.")
mm.moveToHome(axis)
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 v1. ###

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure actuators
axesToMove = [1,2,3]
for axis in axesToMove:
    mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.timing_belt_150mm_turn)

### HOMING ###

# Home actuators before performing absolute moves
print("All Axes Moving Home Sequentially")
mm.moveToHomeAll()
print("All Axes homed.")

### SIMULTANEOUS ABSOLUTE MOVES OF THREE AXES ###

#   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")
```

readEncoder.py

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

### This Python example showcases how to read encoder positions with MachineMotion v1. ###

mm = MachineMotion()

# Adjust this line to match whichever AUX port the encoder is plugged into
encoderPort = AUX_PORTS.aux_1

checkInput = True
print("Reading and printing encoder output for 10 seconds:")
for i in range(1,30):
   realTimeOutput = mm.readEncoder(encoderPort, ENCODER_TYPE.real_time)
   stableOutput = mm.readEncoder(encoderPort, ENCODER_TYPE.stable)
   print("Encoder on port AUX " + str(encoderPort+1) + "\t Realtime Position =" + str(realTimeOutput) + " counts\t StablePosition = " + str(sta
   time.sleep(.25)

   if realTimeOutput != 0:
      checkInput = False

if(checkInput):
   print("The encoder is not receiving any data. Please check the following: \n\t Is the encoder plugged into AUX " + str(encoderPort+1) + "?
```

setAcceleration.py

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

### This Python example configures system acceleration for MachineMotion v1. ###

mm = MachineMotion()

acceleration = 500      # The max acceleration [mm/s^2] that all subsequent moves will move at
mm.setAcceleration(acceleration)
print("Global acceleration set to " + str(acceleration) + "mm/s^2.")
```

setPosition.py

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

### This Python example showcases how to set actuator position with MachineMotion v1. ###

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = 1
mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.rack_pinion_mm_turn)

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

setSpeed.py

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

### This Python example configures system speed for MachineMotion v1. ###

mm = MachineMotion()

speed = 500      # The max speed [mm/s] that all subsequent moves will move at
mm.setSpeed(speed)
print("Global speed set to " + str(speed) + "mm/s.")
```

stopAllMotion.py

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

### This Python example showcases how to stop motion on MachineMotion v1. ###

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = 1
mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.rack_pinion_mm_turn)

# Configure Move Parameters
speed = 200
mm.setSpeed(speed)

# 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.")
```

waitForMotionCompletion.py

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

### This Python example showcases how to wait for actuator motion completion on MachineMotion v1. ###

mm = MachineMotion()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = 1
mm.configAxis(axis, MICRO_STEPS.ustep_8, MECH_GAIN.rack_pinion_mm_turn)

# 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")
```

configHomingSpeed.py (V2)

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

### This Python example configures homing speed for MachineMotion v2. ###

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

### Configuring ###

axes = [1,2]
homingSpeeds = [50,100] # The homing speeds to set for each axis, in mm/sec

mm.configServo(1, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)
mm.configServo(2, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)
mm.configHomingSpeed(axes, homingSpeeds)    # Sets homing speeds for all selected axes.

### Testing the configuration ###

axis = 1    # The axis to move

print("Moving axis " + str(axis) + " by 100mm.")
mm.moveRelative(axis, 100)
mm.waitForMotionCompletion()

#Homes the axis at the newly configured homing speed.
print("Homing axis " + str(axis))

mm.moveToHome(axis)
mm.waitForMotionCompletion()
```

configMultiDriveServo.py (V2)

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

### This Python example configures and moves a multi-drive axis. ###

### Multiple motors that are mechanically linked can now be treated
### as though they are a single axis. Once propely configured,
### multi-drive axes can be moved just as with single-drive axes.

### Machine Motion declaration
mm = MachineMotionV2()

### If not on a conveyor, the parent drive must have home and end sensors.
### The child drive's sensors will be ignored.
parent = 1
parentDirection = DIRECTION.NORMAL
child = 2
childDirection = DIRECTION.NORMAL

mechGain = MECH_GAIN.timing_belt_150mm_turn
motorCurrent = 5.0 # Current (A)

print("--> Configuring parent drive " + str(parent) + " with child " + str(child))

### Configure your multi-drive axis by passing the list of drives, directions and parent drive.
mm.configServo([parent, child], mechGain, [parentDirection, childDirection], motorCurrent, parentDrive = parent)

### The parent and child drives are now linked.
### Control the multi-drive axis via the parent drive:

print("--> Multi-drive axis is moving relative by 200mm!")
mm.moveRelative(parent, 200)
mm.waitForMotionCompletion()

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

configStepperServo.py (V2)

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

### This Python example configures actuators for a MachineMotion v2. ###

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure a timing belt actuator in servo mode, on drive 1
drive = 1
mechGain = MECH_GAIN.timing_belt_150mm_turn
direction = DIRECTION.NORMAL
motorCurrent = 5.0 # Amperes
mm.configServo(drive, mechGain, direction, motorCurrent, tuningProfile=TUNING_PROFILES.DEFAULT)
# If you have a different size motor:
# mm.configServo(drive, mechGain, direction, motorCurrent, motorSize=MOTOR_SIZE.SMALL)

# Configure an electric cylinder actuator in stepper mode, on drive 2
drive = 2
mechGain = MECH_GAIN.electric_cylinder_mm_turn
direction = DIRECTION.NORMAL
motorCurrent = 1.6 # Amperes
mm.configServo(drive, mechGain, direction, motorCurrent, tuningProfile=TUNING_PROFILES.DEFAULT) # Microsteps will default to 8.
# If you need to use a different microstepping :
#mm.configStepper(drive, mechGain, direction, motorCurrent, microSteps = MICRO_STEPS.ustep_4)
```

controlBrakes.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# 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)
```

digitalRead.py (V2)

```
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 (V2)

```
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)
```

eStop.py (V2)

```
#!/usr/bin/python
import sys, time
sys.path.append("../..")
from MachineMotion import *

### This Python example showcases how to control eStop on MachineMotion v2. ###

### MachineMotion CONFIGURATION ###

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

# Define a callback to process estop status
def templateCallback(estop_status):
    print("eStop status is : " + str(estop_status))
    if estop_status == True :
        # executed when you enter estop
        print("MachineMotion is in estop state.")
    elif estop_status == False :
        # executed when you exit estop
        print("MachineMotion is not in estop state.")
mm.bindeStopEvent(templateCallback)

### ESTOP TRIGGER ###

result = mm.triggerEstop()
if result == True   : print("--> Software stop triggered")
else                : print("--> Failed to trigger software stop")

time.sleep(3.0)

### ESTOP RELEASE ###

result = mm.releaseEstop()
if result == True   : print("--> Software stop released")
else                : print("--> Failed to release software stop")

time.sleep(3.0)

### SYSTEM RESET ###

result = mm.resetSystem()
if result == True   : print("--> System has been reset")
else                : print("--> Failed to reset system")

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

getEndStopState.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# 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) )
```

getPositions.py (V2)

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

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

###### CONFIGURING MACHINEMOTION ######

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure actuator
axis = 1
print("--> Configuring actuator")
mm.configServo(axis, MECH_GAIN.rack_pinion_mm_turn, DIRECTION.POSITIVE, 5.0)

# 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")
```

moveContinuous.py (V2)

```
#!/usr/bin/python
import sys, time
sys.path.append("../..")
from MachineMotion import *

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

### MachineMotion CONFIGURATION ###

# Create MachineMotion instance
mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure actuator
conveyor_axis = 1
mm.configServo(conveyor_axis, MECH_GAIN.roller_conveyor_mm_turn, DIRECTION.NORMAL, 5.0)

### CONTINUOUS MOVES ###

# Start the continuous move
print("Start Conveyor Move...")
print("Continuous move: speed 100mm/s & acceleration 50mm/s^2")
mm.moveContinuous(conveyor_axis, 100, 50)
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)

# Stop the continuous move
print("Stop Conveyor Move...")
mm.stopMoveContinuous(conveyor_axis, 500)
time.sleep(3)
print("--> Example completed")
```

moveRelative.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = 1
mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)

# 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.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure actuators
axesToMove = [1,2,3]
for axis in axesToMove:
    mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)

# 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")
```

moveToHome.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = AXIS_NUMBER.DRIVE1
mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)

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

moveToHomeAll.py (V2)

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

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

### MachineMotion configuration ###

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

### Home all axes sequentially
print ("All Axes Moving Home sequentially")
mm.moveToHomeAll()
mm.waitForMotionCompletion()
print("All Axes Homed")
```

moveToPosition.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Axis configuration
axis = 1            #The axis that you'd like to move
mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)

# 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 (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure actuators
axesToMove = [1,2,3]
for axis in axesToMove:
    mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)

# 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")
```

oneDriveControl.py (V2)

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

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

### MachineMotion configuration ###

mm = MachineMotionV2OneDrive()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

axis = AXIS_NUMBER.DRIVE1
motorCurrent = 7.5 # current (A)

print("--> Configuring axis " + str(axis) + " as timing belt")
mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, motorCurrent)

### 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!")
```

powerSwitch.py (V2)

```
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 (V2)

```
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)
```

setAcceleration.py (V2)

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

### This Python example configures system acceleration for MachineMotion v2. ###

mm = MachineMotionV2()

acceleration = 500      # The max acceleration [mm/s^2] that all subsequent moves will move at
mm.setAcceleration(acceleration)
print("Global acceleration set to " + str(acceleration) + "mm/s^2.")
```

setPosition.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = 1
mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)

# 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")
```

setSpeed.py (V2)

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

### This Python example configures system speed for MachineMotion v2. ###

mm = MachineMotionV2()

speed = 500      # The max speed [mm/s] that all subsequent moves will move at
mm.setSpeed(speed)
print("Global speed set to " + str(speed) + "mm/s.")
```

stopAllMotion.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = 1
mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)

# Configure Move Parameters
speed = 200
mm.setSpeed(speed)

# 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.")
```

waitForMotionCompletion.py (V2)

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

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

mm = MachineMotionV2()

# When starting a program, one must remove the software stop before moving
print("--> Removing software stop")
mm.releaseEstop()
print("--> Resetting system")
mm.resetSystem()

# Configure the actuator
axis = 1
mm.configServo(axis, MECH_GAIN.timing_belt_150mm_turn, DIRECTION.NORMAL, 5.0)

# 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")
```

back to API Reference