ftdu documentation

Author

Feb 16, 2019

Contents

Contents:

CHAPTER 1

ftdu package

1.1 Module contents

Python client to communicate with a ftDuino via USB.

It uses the `ftduino_direct` sketch written by Peter Habermehl. See <https://github.com/PeterDHabermehl/ftduino_direct>

class ftdu.**BaseFtDuino**(*path=None*)

Base class to communicate with a ftDuino.

This class implements all high level functions of the ftDuino API.

To issue other commands, the *comm()* method can be used.

close()

Closes the connection to the ftDuino.

Use the `with` statement to ensure that this method is called.

```
with BaseFtduino() as ftd:
    ftd.led = True
```

comm(*cmd*)

Low level access to the ftDuino.

See <https://github.com/PeterDHabermehl/ftduino_direct#use> for a list of commands.

Parameters **cmd** – The command to execute.

Return type str

Returns The result of the command or `None` in case of an error.

counter_clear(*port*)

Clears the provided counter port.

Parameters **port** – Port name, i.e. 'C1'. The port name is case-insensitive.

counter_get(*port*)
Returns the value of the provided counter port.

Parameters **port** – Port name, i.e. 'C1'. The port name is case-insensitive.

Return type int

Returns The value of the provided counter.

counter_get_state(*port*)
Returns the state of the provied counter port.

Parameters **port** – Port name, i.e. 'C1'. The port name is case-insensitive.

Return type bool

Returns The state, a boolean of the port.

counter_set_mode(*port*, *mode*)
Sets the mode of the provided counter.

Parameters

- **port** – Port name, i.e. 'C1'. The port name is case-insensitive.
- **mode** – 'none', 'rising', 'falling', or 'any' (case-insensitive)

ftduino_direct_get_version()
Returns the ftduino_direct version

Returns A version string.

ftduino_id_get()
Returns the ID of the connected ftDuino.

Returns The ID of the ftDuino.

ftduino_id_set(*identifier*)
Sets the ftDuino ID.

Parameters **identifier** (*str|unicode*) – The identifier.

input_get(*port*)
Reads a value from the provided input port.

Parameters **port** – Port name, i.e. 'I1'. The port name is case-insensitive.

Return type int

Returns The integer value read from the specified port.

Raise ValueError in case of an error.

input_set_mode(*port*, *mode*)
Sets the mode for the provided input port.

Parameters

- **port** – Port name, i.e. 'I1'. The port name is case-insensitive.
- **mode** – 'switch', 'resistance', or 'voltage' (case-insensitive), see constants `ftdu.INPUT_MODE_SWITCH`, `ftdu.INPUT_MODE_RESISTANCE` and `ftdu.INPUT_MODE_VOLTAGE`.

Raise ValueError in case the provided mode is unknown.

led_set(*enable*)
Switches the LED on or off

Parameters **enable** – `True` to switch the LED on, `False` to switch the LED off.

motor_counter(*port*, *mode*, *pwm*, *counter*)
Sets the state of an encoder motor.

Parameters

- **port** – Port name, i.e. 'M1'. The port name is case-insensitive.
- **mode** – 'off', 'left', 'right', or 'brake' (case-insensitive), see constants `ftdu.MOTOR_OFF`, `ftdu.MOTOR_LEFT`, `ftdu.MOTOR_RIGHT`, and `ftdu.MOTOR_BRAKE`.
- **pwm** – Pulse-width modulation value.
- **counter** – Counter value. The motor stops after reaching the value.

motor_counter_active(*port*)
Returns if a counter is active for the given motor port.

Parameters **port** – Port name, i.e. 'M1'. The port name is case-insensitive.

Returns `True` if the counter is active, otherwise `False`.

motor_counter_set_brake(*port*, *enable*)
Indicates if the motor should be stopped indirectly (`False`) or directly (`True`).

Parameters

- **port** – Port name, i.e. 'M1'. The port name is case-insensitive.
- **enable** – `True` to set the brake, otherwise `False`

motor_set(*port*, *mode*, *pwm=None*)
Sets the provided motor port into the given state.

Parameters

- **port** – Port name, i.e. 'M1'. The port name is case-insensitive.
- **mode** – 'off', 'left', 'right', or 'brake' (case-insensitive), see constants `ftdu.MOTOR_OFF`, `ftdu.MOTOR_LEFT`, `ftdu.MOTOR_RIGHT`, and `ftdu.MOTOR_BRAKE`.
- **pwm** – Pulse-width modulation value. If `None` the max. PWM value will be used.

output_set(*port*, *mode*, *pwm=None*)
Sets the provided output port into the provided mode.

Parameters

- **port** – Port name, i.e. 'O1'. The port name is case-insensitive.
- **mode** – 0 = OFF, 1 = HIGH, 2 = LOW
- **pwm** – Pulse-width modulation value. If `None` the value depends on the mode. If the mode is `1` (high), the pwm will be set to the max. pwm value, otherwise to the min. pwm value.

ultrasonic_enable(*enable*)
Enables / disables the ultrasonic sensor.

Parameters **enable** (*bool*) – `True` to enable, `False` to disable.

ultrasonic_get()
Reads and returns the value of the ultrasonic sensor.

Return type int

Returns The value of the ultrasonic sensor.

class ftdu.**FtDuino**(*path=None*)

This class provides all functions of the *BaseFtDuino* and adds a higher level API to access ports via attributes.

The red LED can be switched on and off via `led = True` or `led = False`.

```
ftd = FtDuino()
ftd.led = True  # Switches the LED on.
```

The input ports can be read by using the port names (i1 .. i8), i.e. `ftd.i1` to get the value of input port "I1".

The output ports (o1 .. o8) can be enabled / disabled by assigning a boolean value.

Example:

```
ftd = FtDuino()
ftd.o1 = True   # Sets the output O1 to HIGH with a max. PWM value
ftd.o2 = False  # Sets the output O2 to LOW with a min. PWM value
```

Further, it is possible to specify the PWM value if a tuple is used:

```
ftd = FtDuino()
ftd.o1 = ftdu.HIGH, ftdu.MAX / 2  # Sets the output O1 to HIGH with half speed
```

This class provides also methods to control motors at the ports M1 .. M4.

```
ftd = FtDuino()
ftd.m1_left()   # Rotation left at full speed
ftd.m1_right(pwm=ftdu.MAX / 2)  # Rotation right with half speed

# Rotation right, full speed, stop after 38 steps (encoder motor)
ftd.m2_right(steps=38)
```

c1_clear()

Clears counter C1 (sets the counter value to zero).

c2_clear()

Clears counter C2 (sets the counter value to zero).

c3_clear()

Clears counter C3 (sets the counter value to zero).

c4_clear()

Clears counter C4 (sets the counter value to zero).

m1_brake(*pwm=None*, *steps=None*)

Brakes the motor at M1.

See also `motor_counter_set_brake()`

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m1_left(*pwm=None*, *steps=None*)

Sets the rotation of the motor at M1 to "left".

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.

- **steps** – Number of steps until the motor stops (encoder motor required).

m1_off(*steps=None*)
Switches the motor at M1 off.

Parameters **steps** – Number of steps until the motor stops (encoder motor required).

m1_right(*pwm=None*, *steps=None*)
Sets the rotation of the motor at M1 to "right".

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m2_brake(*pwm=None*, *steps=None*)
Brakes the motor at M2.

See also `motor_counter_set_brake()`

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m2_left(*pwm=None*, *steps=None*)
Sets the rotation of the motor at M2 to "left".

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m2_off(*steps=None*)
Switches the motor at M2 off.

Parameters **steps** – Number of steps until the motor stops (encoder motor required).

m2_right(*pwm=None*, *steps=None*)
Sets the rotation of the motor at M2 to "right".

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m3_brake(*pwm=None*, *steps=None*)
Brakes the motor at M3.

See also `motor_counter_set_brake()`

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m3_left(*pwm=None*, *steps=None*)
Sets the rotation of the motor at M3 to "left".

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m3_off(*steps=None*)
Switches the motor at M3 off.

Parameters **steps** – Number of steps until the motor stops (encoder motor required).

m3_right(*pwm=None*, *steps=None*)
Sets the rotation of the motor at M3 to "right".

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m4_brake(*pwm=None*, *steps=None*)
Brakes the motor at M4.

See also `motor_counter_set_brake()`

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m4_left(*pwm=None*, *steps=None*)
Sets the rotation of the motor at M4 to "left".

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

m4_off(*steps=None*)
Switches the motor at M1 off.

Parameters **steps** – Number of steps until the motor stops (encoder motor required).

m4_right(*pwm=None*, *steps=None*)
Sets the rotation of the motor at M4 to "right".

Parameters

- **pwm** – Pulse-width modulation value. If `None` the value is set to the maximum.
- **steps** – Number of steps until the motor stops (encoder motor required).

c1
Returns the value of counter "C1".

Return type int

Returns The value of counter "C1".

c1_state
Returns the state of counter "C1".

Return type bool

Returns `True` if the counter is active, otherwise `False`.

c2
Returns the value of counter "C2".

Return type int

Returns The value of counter "C2".

c2_state
Returns the state of counter “C2”.

Return type bool

Returns `True` if the counter is active, otherwise `False`.

c3
Returns the value of counter “C3”.

Return type int

Returns The value of counter “C3”.

c3_state
Returns the state of counter “C3”.

Return type bool

Returns `True` if the counter is active, otherwise `False`.

c4
Returns the value of counter “C4”.

Return type int

Returns The value of counter “C4”.

c4_state
Returns the state of counter “C4”.

Return type bool

Returns `True` if the counter is active, otherwise `False`.

i1
Returns the value of input port “I1”.

Return type int

Returns The value of input port “I1”.

i2
Returns the value of input port “I2”.

Return type int

Returns The value of input port “I2”.

i3
Returns the value of input port “I3”.

Return type int

Returns The value of input port “I3”.

i4
Returns the value of input port “I4”.

Return type int

Returns The value of input port “I4”.

i5
Returns the value of input port “I5”.

Return type int

Returns The value of input port “I5”.

i6
Returns the value of input port “I6”.

Return type int

Returns The value of input port “I6”.

i7
Returns the value of input port “I7”.

Return type int

Returns The value of input port “I7”.

i8
Returns the value of input port “I8”.

Return type int

Returns The value of input port “I8”.

m1_counter_active
Returns if the motor counter for port M1 is active.

Returns `True` if active, `False` otherwise.

m2_counter_active
Returns if the motor counter for port M2 is active.

Returns `True` if active, `False` otherwise.

m3_counter_active
Returns if the motor counter for port M3 is active.

Returns `True` if active, `False` otherwise.

m4_counter_active
Returns if the motor counter for port M4 is active.

Returns `True` if active, `False` otherwise.

ultrasonic
Returns the value of the ultrasonic sensor.

Return type int

Returns The value of the ultrasonic sensor.

ftdu.**ftduino_find_by_name**(*name*)
Returns the path of the ftDuino with the specified *name*.

Parameters **name** – Name of the ftDuino.

Returns The path of the ftDuino or `None` if the ftDuino was not found.

ftdu.**ftduino_iter**()
Returns an iterator / generator over all ftDuinos connected to the host device.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

U