# serializer.py



NOTE: See the offical SerializerTM manual at:
http://www.roboticsconnection.com/multimedia/docs/Serializer_3.0_UserGuide.pdf

## Modules

math serial threading

## Classes

GP2D12
PhidgetsCurrent
PhidgetsTemperature
PhidgetsVoltage
Ping
Serializer

### class GP2D12

Methods defined here:

**__init__**(self, serializer, pin)

```
Usage: myIR = GP2D12(serializer, pin)
       reading = myIR.value()
       reading = myIR.value(cached=True) # gets value from the cache
```

The Sharp GPD12 IR Sensor class wraps an analog sensor port and converts the raw sensor reading to either inches or cm depending on the units settings.

**value**(self, cached=False)

### class PhidgetsCurrent

Methods defined here:

**__init__**(self, serializer, pin, model=20, ac_dc='dc')

```
Usage: myAmps = PhidgetsCurrent(serializer, pin)
       reading = myAmps.value()
       reading = myAmps.value(cached=True) # gets value from the cache
```

The PhidgetsCurrent class wraps an analog sensor port and converts the raw sensor reading to either inches or cm depending on the units settings.

**value**(self, cached=False)

### class PhidgetsTemperature

Methods defined here:

**__init__**(self, serializer, pin, units='F')

```
Usage: myTemp = PhidgetsTemperature(serializer, pin)
       reading = myTemp.value()
       reading = myTemp.value(cached=True) # gets value from the cache
```

The Phidgets Temperature Sensor class wraps an analog sensor port and converts the raw sensor reading to either Farhenheit or Celcius depending on the units argument.

**value**(self, cached=False)

**value**(self, cached=False)

## class **PhidgetsVoltage**

Methods defined here:

**__init__**(self, serializer, pin)

```
Usage: myVolts = PhidgetsVoltage(serializer, pin)
       reading = myVolts.value()
       reading = myVolts.value(cached=True) # gets value from the cache
The Phidgets Voltage Sensor class wraps an analog sensor port and converts the raw
sensor reading to volts.
```

**value**(self, cached=False)

## class **Ping**

Methods defined here:

**__init__**(self, serializer, pin)

```
Usage: myPing = Ping(serializer, pin)
       reading = myPing.value()
       reading = myPing.value(cached=True) # gets value from the cache
The Parallax Ping Sonar Sensor class wraps a digital sensor port returns the value
from the pping() command which in turn is in inches or cm depending on the units settings.
```

**value**(self, cached=False)

## class **Serializer**

Methods defined here:

**__init__**(self, port='COM12', baudrate=19200, timeout=5)

**blink_led**(self, id, rate)

```
Usage 1. blink_led(id, rate)
Usage 2. blink_led([id1, id2], [rate1, rate2])
The blink_led command can blink one of the two onboard green LEDs
simultaneously, or individually. Each complex parameter is comprised
of an <ledId:blinkRate> pair. The ledId specifies which of the two
green LEDs to blink, and blinkRate specifies the delay between blinks.
The minimum blink rate is 1, and the largest is 127. A value of 0 turns
the led off.
```

**clear_encoder**(self, id)

```
Usage 1: clear_encoder(id)
Usage 2: clear_encoder([id1, id2])
The clear_encoder command clears the values of the encoder
count (channel B) for the specified encoder Id.
```

**close**(self)

```
Close the serial port.
```

**connect**(self)

**digo**(self, id, dist, vel)

```
Usage 1: m(id, dist, vel)
Usage 2: digo([id1, id2], [dist1, dist2], [vel1, vel2])
Simply put, the digo command allows you to command your robot to
travel a specified distance, at a specified speed. This command uses
the internal VPID and DPID algorithms to control velocity and distance.
Therefore, you must have dual motors, and dual wheel encoders
connected to the Serializer motor ports and encoder inputs.
```

**execute**(self, cmd)
Thread safe execution of "cmd" on the SerializerTM returning a single value.

**execute_ack**(self, cmd)
Thread safe execution of "cmd" on the SerializerTM returning True if response is ACK.

**execute_array**(self, cmd)
Thread safe execution of "cmd" on the SerializerTM returning an array.

**execute_int**(self, cmd)
Thread safe execution of "cmd" on the SerializerTM returning an int.

**fw**(self)
The fw command returns the current firmware version.

**get_all_analog**(self)
Return the readings from all analog ports.

**get_baud**(self)
Get the current baud rate on the serial port.

**get_compass**(self, i2c_addr=None)
Usage 1. heading = get_compass()
Usage 2. heading = get_compass(i2c_addr)
The get_compass command queries a Devantech CMPS03 Electronic
compass module attached to the Serializers I2C port.
The current heading is returned in Binary Radians, or BRADS.To
convert BRADS to DEGREES, multiply BRADS by 360/255 (~1.41).
The default I2C address is 0xC0, however another I2C address can be
supplied as an optional parameter.

**get_dpid**(self)
Get the PIDA parameter values.

**get_encoder**(self)
The get_encoder command returns the encoder type for the SerializerTM,
single (0) or quadrature (1).

**get_encoder_count**(self, id)
Usage 1: get_encoder_count(id)
Usage 2: get_encoder_count([id1, id2])
The get_encoder_count command returns the values of the encoder
count (channel B) for the specified encoder Id(s). NOTE: The encoder
counts for channel A are used for internal VPID and DPID algorithms.

**get_encoder_resoluton**(self)

**get_gear_reduction**(self)

**get_io**(self, id)
Usage 1: get_io(id)
Usage 2: get_io([id1, id2, id3, ... , idN])
The get_io command changes the pin, pinId (range 0-12), to an input
(if it was an output), and gets the value of the specified General
Purpose I/O lines on the SerializerTM. The valid range of I/O pin Ids is
0 thru 12. More than one pid can be specified by enter a list as argument.

**get_maxez1**(self, triggerPin, outputPin)
The maxez1 command queries a Maxbotix MaxSonar-EZ1 sonar
sensor connected to the General Purpose I/O lines, triggerPin, and
outputPin, for a distance, and returns it in Centimeters. NOTE: MAKE
SURE there's nothing directly in front of the MaxSonar-EZ1 upon
power up, otherwise it wont range correctly for object less than 6
inches away! The sensor reading defaults to use English units
(inches). The sensor distance resolution is integer based. Also, the
maxsonar trigger pin is RX, and the echo pin is PW.

**get_pids**(self)

Once a digo command is issued, an internal state variable within the
firmware is set to 1, and it stays in that state until the algorithm has
completed. Upon completion, the state is set to 0. The pids
command simply returns the value of the internal variable to
determine if the algorithms is currently busy, or if it has finished, thus
allowing subsequent digo commands to be issued w/o clobbering
previous ones.

**get_units**(self)

The get_units returns the current units used for sensor
readings. Values are 0 for metric mode, 1 for English mode, and 2 for
raw mode. In raw mode, srf04, srf05, pping, and maxez1 return
reading in units of 0.4us. srf08 and srf10 return readings of 1us..

**get_vpid**(self)

Get the PIDL parameter values.

**get_wheel_diameter**(self)

**get_wheel_track**(self)

**i2c**(self, op, addr, data=None)

Usage1: i2c(op, addr)
Usage2: i2c(op, addr, data)
The flexible i2c command allows you to execute a generic i2c read, or
write command to the specified device at address addr. Depending on
whether you issue a read or write, additional parameters vary.

**line**(self, addr, newaddr=None, seven=False)

Queries a RoboticsConnection Line Following Sensor at address addr.
If the -a option is specified, then the address of the module will be
changed to the new address associated w the -a switch.
If the optional 7 is appended to the end of the line command, e.g.
line7, then two additional values will be returned from those Line
Following Sensors (manufactured after 11/1/07) which have additional
sensor inputs on the sides of the board. This can be used to read
additional Single Line Following sensors, or read any type of on/off
momentary switch, such those used for bumpers.

**mogo**(self, id, vel)

Usage 1: mogo(id, vel)
Usage 2: mogo([id1, id2], [vel1, vel2])
The mogo command sets motor speed using one or more complex
parameters containing a <motorId:spd> value pair.
The motorId can be either 1 or 2, which corresponds to the Motor
Terminal port.
The vel value specifies the motor velocity, and it's range depends on
your VPID settings. See the VPID parameters section below to
determine your MAX velocity. A positive value rotates the motors in
one direction, which a negative value rotates the motors in the
opposite direction.
You will have to determine which direction is positive for your motors,
and connect the motors wires to the terminals on the Serializer board
in the appropriate configuration.

**open**(self)

Open the serial port.

**pping**(self, pinId)

The srf05/Ping command queries an SRF05/Ping sonar sensor
connected to the General Purpose I/O line pinId for a distance,
and returns it in the units configured (default is English - inches).
If the Serializer units are configured (using cfg units) for raw mode,
pping and srf05 return readings in units of 0.4us, and the max
distance returned is 65000 (out of range). When configured for
English units, max distance returned is 100 inches (out of range), and
when configured for Metric units, max distance returned is 255 (out of
range). Sonar distance resolution is integer based.

**pwm**(self, id, vel, rate=None)

Usage 1: pwm(id, vel)
Usage 2: pwm(id, vel, rate=r)
Usage 3: pwm([id1, id2], [vel1, vel2])

Usage 4: pwm([id1, id2], [vel1, vel2], rate=r)
The pwm command sets the Pulse Width Modulation value for
Motor 1 & Motor 2. Each complex parameter is a motor
<motorId:pwm value> pair, where the motor id can be 1 or 2, and the
pwm value can be -100 to 100. Each complex parameter pair is
separated by one or more spaces.
The optional rate parameter allows the
motor(s) speed(s)s to be ramped up or down to the specified speed
from the current motor speed.

**recv**(self)
This command should not be used on its own: it is called by the execute commands
below in a thread safe manner.

**recv_ack**(self)
This command should not be used on its own: it is called by the execute commands
below in a thread safe manner.

**recv_array**(self)
This command should not be used on its own: it is called by the execute commands
below in a thread safe manner.

**recv_int**(self)
This command should not be used on its own: it is called by the execute commands
below in a thread safe manner.

**reset**(self)
The reset command resets the SerializerTM board and reboots it. You
will see the SerializerTM welcome screen appear after a short delay.
Once the welcome string appears, the SerializerTM is ready to accept
commands.

**restore**(self)
Restores the factory default settings, and resets the board. NOTE:
This will erase any configurations you have saved to EEPROM,
including VPID, DPID, and baud rate settings.

**rotate**(self, angle, vel)

**send**(self, cmd)
This command should not be used on its own: it is called by the execute commands
below in a thread safe manner.

**sensor**(self, id)
Usage 1: reading = sensor(id)
Usage 2: readings = sensor([id1, id2, ..., idN])
The sensor command returns the raw A/D (8 bit) reading from the
analog sensor ports 0-5. Multiple values can be read at a time by
specifying multiple pins as a list. Pin 5 is 1/3 of
the voltage of the power supply for the SerializerTM. To calculate the
battery voltage, simply multiply the value returned by Sensor 5 by
15/1028.

**servo**(self, id, pos)
Usage 1: servo(id, pos)
Usage 2: servo([id1, id2, ... , idN], [pos1, pos2, ..., posN)
The servo command sets a servo connected to General Purpose I/O
port the specified position. The value of the position can range from
-99 to 100, where 0 is the center position. Setting the position to -100
will disable the servo, allowing it to turn freely by hand.
Each parameter is a <servo id:position> pair, where the servo
id can be 1,2,3,4,5, or 6.

**set_baud**(self, baudrate)
The set_baud command configures the serial baud rate on the
SerializerTM. Values can be 0=2400, 1=4800, 2=9600, 3=19200,
4=57600, or 5=115200. You can also type in the actual baud rate
string as well (e.g. 19200). The default baud rate used to
communicate with the Serializer is 19200. The cfg baud command
without a parameter returns the value currently stored in EEPROM.

**set_dpid**(self, prop, integ, deriv, accel)
```
The set_dpid command gets/sets the PIDA (Proportional, Integral,
Derivative, and Acceleration) parameters for the distance PID control
on the SerializerTM. If the PIDA parameters are absent, the PIDA
values are returned. Otherwise the PIDA parameters are parsed, and
saved (in eeprom).
```

**set_encoder**(self, encoder_type)
```
The set_encoder command configures the internal encoder type to be either
single (0) or quadrature (1) type. This information is saved in the
EEPROM, so that the configuration will be retained after a reboot. If
you are using a quadrature encoder (dual channels), and the Serializer
is configured for single encoder operation, then the second quadrature
channel will be ignored. Thus make sure the correct encoder type is
configured according to your setup. The cfg enc command without a
parameter returns the value currently stored in EEPROM.
```

**set_encoder_resolution**(self, ticks)

**set_gear_reduction**(self, ratio)

**set_io**(self, id, val)
```
Usage 1: set_io(id, val)
Usage 2: set_io([id1, id2, ... , idN], [val1, val2, ..., valN)
The set_io command sets the specified General Purpose I/O line pinId
(range 0-12) on the SerializerTM to the specified value. Each complex
parameter is a <pinId:value> pair, where the valid range of pinId is 0
thru 12, and value can be 0 or 1 which corresponds to 0v or +5V
respectively.  Also I/O lines 4,5,6,7, 8 and 9 cannot be used if you
have servos connected to them. Pin 10, 11, and 12 correspond to the
internal h-bridge enable, SCL, and SDA respectively.
```

**set_rpid**(self, r)
```
The rpid command sets the default PID params known to work with
either the Stinger or Traxster Robotic Kits in the firmware. This makes
it quick and easy to set up the PID params for both robots.
```

**set_units**(self, units)
```
The set_units command sets the internal units used for sensor
readings. Values are 0 for metric mode, 1 for English mode, and 2 for
raw mode.  In raw mode, srf04, srf05, pping, and maxez1 return
reading in units of 0.4us. srf08 and srf10 return readings of 1us.
The cfg units command without a parameter returns the value
currently stored in EEPROM.
```

**set_vpid**(self, prop, integ, deriv, loop)
```
The set_vpid command sets the PIDL (Proportional, Integral,
Derivative, and Loop) parameters for the Velocity PID control on the
SerializerTM. The PIDL parameters are parsed, and saved (in
eeprom). For more information on PIDL control, see the PIDL
configuration section below.  By default the Serializer VPID
parameters are configured to work with our Traxster Robot Kit
```

**set_wheel_diameter**(self, diameter)

**set_wheel_track**(self, track)

**sp03**(self, msg, i2c_addr=None)
```
Usage1: sp03(msg)
Usage2: sp03(msg, ic2_addr)
The sp03 command instructs a Devantech SP03 Speech Synthesizer to
speak the appropriate phrase. If a character representing a number in
the range of 0 to 30, then the SP03 will speak previously programmed
canned phrases. If a phrase is sent, then it will speak the phrase. An
optional I2C address can also be specified. Otherwise, the default I2C
address of 0xC4.
```

**srf04**(self, triggerPin, outputPin)
```
The srf04 command queries an SRF04 sonar sensor
connected to the General Purpose I/O lines triggerPin and outputPin,
for a distance and returns it in the units configured (default is English
```

inches). If the Serializer units are configured (using cfg units) for raw mode, srf04 returns readings in units of 0.4us, and the max distance returned is 65000 (out of range). When configured for English units, max distance returned is 100 inches (out of range), and when configured for Metric units, max distance returned is 255 (out of range). NOTE: Sonar distance resolution is integer based.

**srf05**(self, pinId)
The srf05/Ping command queries an SRF05/Ping sonar sensor connected to the General Purpose I/O line pinId for a distance, and returns it in the units configured (default is English - inches). If the Serializer units are configured (using cfg units) for raw mode, pping and srf05 return readings in units of 0.4us, and the max distance returned is 65000 (out of range). When configured for English units, max distance returned is 100 inches (out of range), and when configured for Metric units, max distance returned is 255 (out of range). Sonar distance resolution is integer based.

**srf08**(self, i2c_addr=None)
Usage1: srf08()
Usage2: srf08(ic2_addr)
The srf08/srf10 command queries a Devantech SRF08/SRF10 sonar sensor at address i2c_addr for a distance reading in the units configured (default is English - inches). The i2cAddr parameter is optional, and defaults to 0xE0 for both sensors. The i2c address can be changed for any i2c module using the i2cp command. Sonar distance resolution is integer based. If the Serializer units are configured (using cfg units) for raw mode, srf08 and srf10 return readings in units of 1us.

**srf10**(self, i2caddr=None)
Usage1: srf10()
Usage2: srf10(ic2_addr)
The srf08/srf10 command queries a Devantech SRF08/SRF10 sonar sensor at address ic2_addr for a distance reading in the units configured (default is English - inches). The ic2_addr parameter is optional, and defaults to 0xE0 for both sensors. The i2c address can be changed for any i2c module using the i2cp command. Sonar distance resolution is integer based. If the Serializer units are configured (using cfg units) for raw mode, srf08 and srf10 return readings in units of 1us.

**step**(self, dir, speed, steps)
The step command is used to step a bipolar stepper motor in direction dir, at the specified speed, for the specified number of steps. The dir parameter specifies a CW or CCW rotational direction, and its value can be either 0 (CCW) or 1(CW). Your specific direction is based on the way that you have your bipolar motor connected to the Serializer. The speed parameter can be a value from 0 to 100. The steps parameter specifies the maximum number of steps to take. A value of 0 means step infinitely. Internally, this number is stored in an unsigned 32 bit variable, so the user can specify a larger number of steps.

**stop**(self)
Stop both motors.

**sweep**(self, id, speed, steps)
The sweep command is used to sweep a bipolar motor, for step number of steps, at speed (0-100), thus providing a sweeping motion. The initial rotational direction of sweep is in the CW direction. Upon initial receipt of the command, the firmware will sweep the motor for 1/2 of the number of steps specified, starting in a CW direction. Once that number of steps has occurred, the sweep direction will change, and subsequent sweeps will rotate for the full amount of steps. Thus, the starting point for the motor is in the middle of each sweep. You may stop the sweep by either issuing a sweep command w a 0 speed, or simply sending a stop command.

**tpa81**(self, i2caddr=None)
Usage1: tpa81()
Usage2: tpa81(ic2_addr)
The tpa81 command queries a Devantech TPA81 thermopile sensor for temperature values. It returns 8 temperature values.

**travel_distance**(self, dist, vel)

**vel**(self)

```
    The vel command returns the left and right wheel velocities. The
    velocity returned is based on the PIDL parameter configuration.
```

**voltage**(self, cached=False)

---

Data and other attributes defined here:

**ENCODER_RESOLUTION** = 624

**GEAR_REDUCTION** = 1

**I2C_READ** = 'r'

**I2C_WRITE** = 'w'

**N_ANALOG_PORTS** = 6

**N_DIGITAL_PORTS** = 12

**WHEEL_DIAMETER** = 5

**WHEEL_TRACK** = 14