Voltage Measurements

Single datapoints

⚠ get_voltage : Measure voltage

Reads voltage from specified channel , and returns the value.

Autorange is enabled for this function, and it automatically selects the appropriate voltage range if either A1, or A2 are specified

parameter	description
Channel	Analog input to measure. A1,A2,A3, MIC, SEN, or IN1
return	Voltage from chosen input

x = p.get_voltage('A1')

```
import eyes17.eyes
p = eyes17.eyes.open()

print ('Voltage between A1 and GND = ',p.get_voltage('A1'))
print (p.get_voltage('A2'))
print (p.get_voltage('A3'))
```

⚠ get_voltage_time : Measure voltage with a timestamp

Reads voltage from specified channel , and returns the timestamp and measured value. Return T,V

```
t, v = p.get_voltage_time()
print(f'Measured V:{v} , at T:{t}')
```

get_average_voltage : Makes and returns average of n readings

```
v = p.get_average_voltage(samples=50) #Keyword argument samples.
default=10 if omitted
print(f'Measured V:{v}')
```

voltmeter_autorange

Takes a few readings and expands/contracts the voltage range till the value is comfortably within the range.

range selection is done by calling the `select_range` function described later.

This function is invoked by the `get_voltage`, and `get_voltage_time` functions and does not need to be called separately.

```
p.voltmeter_autorange()
```

Code Example : Diode IV Characteristics

```
import eyes17.eyes
p = eyes17.eyes.open()

from matplotlib import pyplot as plt

voltage = []
current = []

v = 0.0
while v <= 5.0:
  va = set_pv1(v)
  vd = get_voltage('A1')
  i = (va-vd)/1.0         # current in milli Amps
  voltage.append(vd)
  current.append(i)
  v = v + 0.050    # 50 mV step

xlabel('Voltage')
ylabel('Current')
plot(voltage, current, linewidth = 2)
show()
```

Multiple, Equidistant voltage acquisition: Synchronous routines

These calls initiate an oscilloscope acquisition process, and wait for it to complete before returning the data in the form of arrays.

> **Warning**
>
> These calls are blocking, and are not recommended in interactive graphical programs if you intend to record for periods longer than 100mS

∿ capture1 : Single Channel Oscilloscope

Blocking call that fetches oscilloscope traces from any analog input `A1,A2,A3, MIC, SEN, or IN1`

parameter	description
Channel	Analog input to measure. A1,A2,A3, MIC, SEN, or IN1
ns	Number of samples to fetch. Maximum 2500
tg	Timegap between samples in microseconds. Minimum 1.75uS
return	Arrays X(timestamps),Y(Voltages from chosen input)

x,y = p.capture1('A1',500,10)

```
import eyes17.eyes
p = eyes17.eyes.open()

from matplotlib import pyplot as plt
x,y = p.capture1('A1',500,10)
plot(x,y)
show()
```

🔧 capture_action : Single Channel Oscilloscope with digital state control

Blocking call that records and returns an oscilloscope trace from the specified input channel after executing another command such as SET_LOW,SET_HIGH,FIRE_PULSE etc on SQ1

parameter	description
Channel	Analog input to measure. A1,A2,A3, MIC, SEN, or IN1
ns	Number of samples to fetch. Maximum 2500
tg	Timegap between samples in microseconds. Minimum 1.75uS
*args	SET_LOW : set OD1 low before capture
	SET_HIGH : set OD1 high before capture
	FIRE_PULSE : make a high pulse on OD1 before capture.
	Use keyword argument pulse_width = x,where x = width of the pulse in uS. default width =10uS
	Use keyword argument pulse_type = 'high_true' or 'low_true' to decide type of pulse
	x,y = p.capture_action('A1',2000,1,'FIRE_PULSE',interval = 250) #Output 250uS pulse on OD1 before starting acquisition
	SET_STATE : change Digital output immediately after capture starts.
	Use keyword arguments that will be forwarded to the set_state command
return	Arrays X(timestamps in mS),Y(Voltages from chosen input)

x,y = p.capture_action('A1',500,10, 'SET_LOW')

```
import eyes17.eyes
p = eyes17.eyes.open()

from matplotlib import pyplot as plt
x,y = p.capture_action('A1',2000,1,'SET_LOW') #set OD1 LOW before
starting acquisition
plot(x,y)
show()
```

RL Transient Experiment

```
import eyes17.eyes
p = eyes17.eyes.open()

from matplotlib import pyplot as plt
import time


plt.plot([0,.5], [0,0], color='black')
plt.ylim([-5,5])


p.set_state(OD1=1)                      # OD1 to HIGH
time.sleep(.5)
t,v = p.capture_action('A1', 100, 5, 'SET_LOW')

plt.plot(t,v,linewidth = 2, color = 'red')
plt.show()
```

RC Transient Experiment

```
import eyes17.eyes
p = eyes17.eyes.open()
from matplotlib import pyplot as plt
import time

p.set_state(OD1=0)                      # OD1 to LOW
time.sleep(.5)
t,v = p.capture_action('A1', 100, 5, 'SET_HIGH')
plt.plot(t,v,linewidth = 2, color = 'blue')

p.set_state(OD1=1)                      # OD1 to LOW
time.sleep(.5)
t,v = p.capture_action('A1', 100, 5, 'SET_LOW')

plt.plot(t,v,linewidth = 2, color = 'red')
plt.show()
```

∿∿ capture2 : 2 Channel Oscilloscope

Blocking call that fetches oscilloscope traces from A1,A2,A3,MIC .

parameter	description
ns	Number of samples to fetch. Maximum 2500
tg	Timegap between samples in microseconds. Minimum 1.75uS
TraceOneRemap	Analog input for channel 1. It is connected to A1 by default.Channel 2-4 always reads CH2-MIC
return	Arrays X1(timestamps in mS),Y1(Voltage at A1),X2(timestamps in mS),Y2(Voltage at A2),X3(timestamps in mS)

t,v1,t2,v2 = p.capture2(1000,2)

```
import eyes17.eyes
p = eyes17.eyes.open()

from matplotlib import pyplot as plt

p.set_sine(200)

t,v, tt,vv = p.capture2(500, 20)   # captures A1 and A2

xlabel('Time(mS)')
ylabel('Voltage(V)')
plot([0,10], [0,0], 'black')
ylim([-4,4])

plot(t,v,linewidth = 2, color = 'blue')
plot(tt, vv, linewidth = 2, color = 'red')

show()
```

AC-DC Separating Demonstration

```
import eyes17.eyes
p = eyes17.eyes.open()
from matplotlib import pyplot as plt

set_sqr1(200)
select_range('A1',8)
select_range('A2',8)

t,v, tt,vv = capture2(500, 20)   # captures A1 and A2

plt.xlabel('Time(mS)')
plt.ylabel('Voltage(V)')
plt.plot([0,10], [0,0], 'black')
plt.ylim([-6,6])

plt.plot(t,v,linewidth = 2, color = 'blue')
plt.plot(tt, vv, linewidth = 2, color = 'red')

plt.show()
```

Diode Clamping Experiment

```
import eyes17.eyes
p = eyes17.eyes.open()
from matplotlib import pyplot as plt

p.set_sine(200)
p.set_pv1(1.7)       # will clamp at 2.0 + diode drop

maxV = 8

p.select_range('A1', maxV)
p.select_range('A2', maxV)

t,v, tt,vv = p.capture2(500, 20)   # captures A1 and A2

plt.xlabel('Time(mS)')
plt.ylabel('Voltage(V)')
plt.plot([0,10], [0,0], 'black')
plt.ylim([-maxV, maxV])

plt.plot(t,v,linewidth = 2, color = 'blue', label='Input')
plt.plot(tt, vv, linewidth = 2, color = 'red', label='Clamped')

plt.legend(framealpha=0.5)

plt.show()
```

capture4 : 4 Channel Oscilloscope

Blocking call that fetches oscilloscope traces from A1,A2,A3,MIC .

parameter	description
ns	Number of samples to fetch. Maximum 2500
tg	Timegap between samples in microseconds. Minimum 1.75uS
TraceOneRemap	Analog input for channel 1. It is connected to A1 by default.Channel 2-4 always reads CH2-MIC

parameter	description
return	Arrays X1(timestamps in mS),Y1(Voltage at A1),X2(timestamps in mS),Y2(Voltage at A2),X3(timestamps in mS),Y3(Voltage at A3),X4(timestamps in mS),Y4(Voltage at MIC)

t,v1,v2,v3,v4 = p.capture4(1000,2)

```
from matplotlib import pyplot as plt
I=eyes17.Interface()
x1,y1,x2,y2,x3,y3,x4,y4 = I.capture4(800,1.75)
plot(x1,y1)
plot(x2,y2)
plot(x3,y3)
plot(x4,y4)
show()
```

capture1_hr : Single Channel Oscilloscope.

Blocking call that fetches oscilloscope traces from any analog input `A1,A2,A3, MIC, SEN, or IN1` with higher resolution (12-bit).

parameter	description
Channel	Analog input to measure. A1,A2,A3, MIC, SEN, or IN1
ns	Number of samples to fetch. Maximum 2500
tg	Timegap between samples in microseconds. Minimum 1.75uS
return	Arrays X(timestamps),Y(Voltages from chosen input)

x,y = p.capture1_hr('A1',500,10)

```
import eyes17.eyes
p = eyes17.eyes.open()

from matplotlib import pyplot as plt
x,y = p.capture1_hr('A1',500,10)
plot(x,y)
show()
```

Configuration options

configure_trigger

`configure_trigger(chan, name, voltage, resolution=10, **kwargs)`

configure trigger parameters for capture commands The capture routines will wait till a rising edge of the input signal crosses the specified level. The trigger will timeout within 8mS tops, and capture routines will start regardless.

These settings will not be used if the trigger option in the capture routines are set to False

parameter	description
chan	channel . 0,1,2,3. corresponding to the channels being recorded by the capture routine(not the analog inputs)
name	the name of the channel. 'A1'... 'SEN'
voltage	Voltage level for triggering

Example code for triggering on A1 at 1.1V threshold in single capture mode

```
p.configure_trigger(0,'A1',1.1)
```

Example code for triggering on A2 at 0V threshold in double capture mode

```
p.configure_trigger(1,'A2',0)
```

select_range

`select_range(channel, voltage_range)`

set the gain of the selected channel. Only for A1 and A2 inputs which have programmable gain amplification available.

parameter	description
channel	A1 or A2
voltage_range	choose from analogRanges.keys() [16,8,4,2.5,1.5,1,.5,.25]

Example code for setting A1 full scale range to ± 4 Volts

```
p.select_range('A1',4)
```

Multiple, Equidistant voltage acquisition: Asynchronous routines

This sections lists separate function calls available for

- instructing the hardware to initiate a capture process
- Querying for the number of samples collected at any time
- Fetching the data buffer either intermittently, or when the acquisition is over.

∿ capture_traces: Initialize the Oscilloscope

`capture_traces(num, samples, tg, channel_one_input='A1', CH123SA=0, **kwargs)`

Instruct the device to start recording data from 1 / 2 / 4 channels simultaneously. use `fetch_trace` to retrieve the data.

parameter	description
num	Channels to acquire. ½/4
samples	Total points to store per channel. Maximum 3200 total.
tg	Timegap between two successive samples (in uSec)
channel_one_input	map channel 1 to 'A1' … 'SEN'

parameter	description
**kwargs	
*trigger	Whether or not to trigger the oscilloscope based on the voltage level set by `configure_trigger`

- Channel 1 can be mapped to any analog input `A1,A2,A3, MIC, SEN, or IN1`
- Channel 2 is A2
- Channel 3 is A3
- and Channel 4 is MIC.

oscilloscope_progress: Fetch status of the Oscilloscope

`oscilloscope_progress`

returns the number of samples acquired by the capture routines, and the conversion_done status

parameter	description
return	conversion done(bool) , waiting_for_trigger(bool), samples acquired (number)

fetch_trace: retrieve collected data buffers

`x,y = fetch_trace(channel_number)`

returns the number of samples acquired by the capture routines, and the conversion_done status

parameter	description
channel_number	1 , 2, 3, or 4. For a single channel capture, use fetch_trace(1). for multiple channels, use fetch_trace(1),… fetch_trace(4)
return	X(list of timestamps in uS) , Y (list of voltages in Volts)

Example code for initializing capture, waiting for completion, and reading the data

```
import time
# Init Single Channel Acquisition. 500 samples with 10uS gap each.
p.capture_traces(1, 500, 10, 'A1', trigger=False)
time.sleep(1e-6 * 500 * 10 + .01)
while 1:
        x = p.oscilloscope_progress()
        print('conversion done : %d, triggered:%d ,samples acquired:
%d'%(x[0],x[1],x[2]))
        if x[0]: break
        time.sleep(0.01)
#Read the data.
x, y = p.fetch_trace(1)
```