Xipppy User Manual

Xipppy Open:

.xipppy\_open()

This is a context manager for Xipppy. The preferred way to manage connections to an NIP:

if \_\_name\_\_ == '\_\_main\_\_':  
 with xp.xipppy\_open():

# Run commands

# Not necessary to close Xipppy

Time:

.time()

Return the most recent NIP time, which is the number of clock cycles at 30 kHz (33.3 μs per cycle) that have occurred since the NIP was started. For example:

with xp.xipppy\_open():

print("Time elapsed after NIP start up: %s min " %(xp.time()/30000/60))

>> Time elapsed after NIP start up: 145.5624 min

Trial:

.trial(oper=150, status=None, file\_name\_base=None, auto\_stop\_time=None, auto\_incr=None, incr\_num=None)

Create a trial packet addressed to the operator with id `*oper*`. All other arguments but *oper* are optional, if they aren't specified they will remain unchanged for the operator. 'Enable remote control' must be enabled for input parameters to have an effect. The resulting or current state corresponding to each input parameter is returned for each call. **To query parameters only, just call with no arguments.**  
*oper:* operator id  
*status*: a string which can be: recording, stopped, paused  
*file\_name\_base*: the base filename that the trial will record to  
*auto\_stop\_time*: the time in seconds after which recording will stop  
*auto\_incr*: a boolean that indicates whether auto\_increment is enabled  
*incr\_num*: the value to set the current file increment counter to

E.g.

xp.trial()

>> status ='stopped'

file\_name\_base= 'C:\\Users\\alireza\\Trellis\\dataFiles\\20190419\\datafile',

auto\_stop\_time= 0

auto\_incr\_status= True

incr\_num= 2

Elec list:

.list\_elec(fe\_type="", max\_elecs=256)

The *list\_elec(‘fetype’)* command returns a sequence on a specified frontend type.

*fe\_type*: Frontend types permitted values are *['stim', 'micro', 'nano', 'surf', 'EMG', 'analog']*

*max\_elecs***:**Maximum number of electrodes to return. If there are more than this many, the extras will be omitted

Example*:*

feList = ['stim', 'micro', 'nano', 'surf', 'EMG', 'analog']

elecList=[]

for fetype in feList:  
 elecList.append(xp.list\_elec(fetype))

if len(elecList[-1]) > 0:  
 print('\n{:s} electrode numbers:'.format(fetype))  
 print(elecList[-1])

>> micro electrode numbers:

array('I', [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31])

Front End index:

.get\_fe(elec)

Return the frontend index of the requested electrode.

*elec*: zero indexed electrode

Stream types:

.get\_fe\_streams(elec, max\_streams=32)

Return a list of stream types supported by the given electrode.

*elec*: zero indexed electrode

*max\_streams*: max strings to return

Example:

*xp.get\_fe\_streams(0) # stream types supported by electrode 0*

>> ['raw', 'hi-res', 'lfp', 'spk']

NIP serial number:

.get\_nip\_serial(max\_size=1024)

Return string of nip serial number, eg 'R00244-0006'.

*max\_size*: maximum size of string

NIPexec version:

*.get\_nipexec\_version(max\_size=1024)*

Return string of nipexec version, eg '1.6.1.23'

*max\_size*: maximum size of string

Frontend R number:

*.get\_fe\_version(elec, max\_size=1024)*

Return R number for frontend for given electrode

*elec*: zero indexed electrode

*max\_streams*: max strings to return

# Signal Functions:

The ‘signal’ functions enable and disable data streams on the NIP and retrieves information about which streams are available for a given electrode and which streams are enabled.

Signal status:

.signal(elec, stream\_ty)

Return a bool indicating whether the stream of the given type on the given electrode is selected.

xp.signal(0,'spk') # spike stream status for electrode 0

>> True

More Functions:

.signal\_raw(elec) ≡ .signal(elec,’raw’)

.signal\_lfp(elec) ≡ .signal(elec,’lfp’)

.signal\_spk(elec) ≡ .signal(elec,’spk’)

.signal\_stim(elec) ≡ .signal(elec,’stim’)

Signal select:

signal\_set(elec, stream\_ty, val)

Select or deselect a signal type on a single electrode. Example:

xp.signal\_set(0,’raw’, True) #Select raw signal on electrode 0

More Functions:

.signal\_set\_raw(elec,val) ≡ .signal\_set(elec,’raw’,val)

.signal\_set\_lfp(elec,val) ≡ .signal\_set(elec,’lfp’,val)

.signal\_set\_spk(elec,val) ≡ .signal(elec,’spk’,val)

.signal\_set\_stim(elec,val) ≡ .signal(elec,’stim’,val)

Impedance:

.impedance(channels)

Take a list of channels and return measured impedances on those channels.

xp.impedance([0,1,3]) #measured impedances for electordes 0,1,3

Retrieve Data Functions:

Functions to retrieve data from continuous data streams such as ‘lfp’, ‘raw’, ‘hires’.

Continuous raw:

.cont\_raw(npoints, elecs, start\_timestamp=0)

Retrieve raw data (sampled at 30 kHz).

*npoints*: number of datapoints to retrieve

*elecs*: list of electrodes to sample

*start\_timestamp*: NIP timestamp to start data at, or most recent if 0

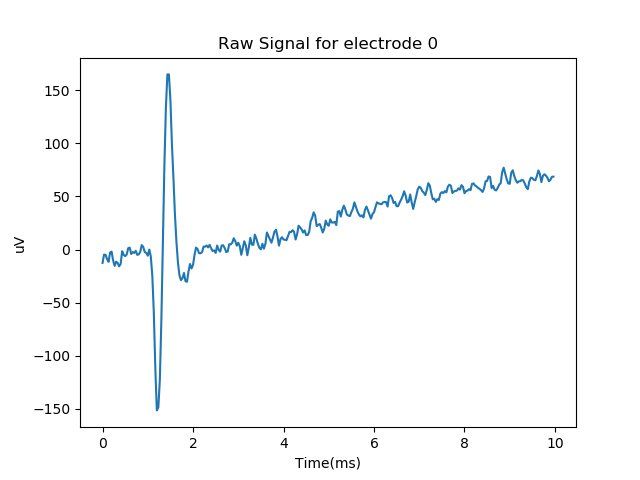
Returns a tuple of (timestamp, data) where the timestamp is that of the first data, and data is a list of data points. If start\_timestamp is not specified, the output timestamp is always 0.

Example:

*import xipppy as xp  
import numpy as np*

*import matplotlib.pyplot as plt*

*if \_\_name\_\_ == '\_\_main\_\_':  
 with xp.xipppy\_open():  
 fs\_clk = 30000  
 xp.signal\_set(0, 'raw', True)  
 elec\_0\_raw = xp.cont\_raw(300, [0], 0)  
 t = np.arange(0, 300000 / fs\_clk, 1000/fs\_clk, dtype=np.float32)  
 plt.plot(t, elec\_0\_raw[0])  
 plt.xlabel('Time(ms)')  
 plt.title('Raw Signal for electrode 0')  
 plt.show()*



Continuous hires:

*.cont\_hires(npoints, elecs, start\_timestamp=0)*

Retrieve hires data (sampled at 2 kHz). Parameters and outputs are the same as the `cont\_raw` function.

Continuous hifreq:

*.cont\_hifreq(npoints, elecs, start\_timestamp=0)*

Retrieve hires data (sampled at 7.5 kHz). Parameters and outputs are the same as the `cont\_raw` function.

Continuous Lfp:

*.cont\_lfp(npoints, elecs, start\_timestamp=0)*

Retrieve lfp data (sampled at 1 kHz). Parameters and outputs are the same as the `cont\_raw` function.

Continuous EMG:

*.cont\_emg(npoints, elecs, start\_timestamp=0)*

Retrieve EMG data. Parameters and outputs are the same as the `cont\_raw` function. If analog I/O is requested, the unfiltered analog data is retrieved, not  
filtered LFP data.

Continuous Mira Status:

*.cont\_status(npoints, elecs, start\_timestamp=0)*

Retrieve Mira status data (2 kHz). Every clock-cycle the Mira front end publishes status information concerning the implant connectivity and sensor information.  
 *| channel | name | units |  
| ------- | ---------------- | ----- |  
| 0 | counter | |  
| 1 | i\_status\_imp | V |  
| 2 | i\_status\_pwr | mA |  
| 3 | adc\_vin\_v | V |  
| 4 | adc\_vin\_a | mA |  
| 5 | adc\_temp\_onboard | C |  
| 6 | adc\_temp\_offboard| C |  
| 7 | pwr\_servo\_state | |  
| 8 | impl\_serial | |  
| 9 | impl\_deviceid | |  
| 10 | impl\_temp | C |  
| 11 | impl\_humidity | % |  
| 12 | impl\_voltage | V |  
| 13 | impl\_ver\_hw | |  
| 14 | impl\_ver\_fw | |*

Spike data:

.spk\_data(elec, max\_spk=1024)

Retrieves spike data. Circular buffer of 1024 spike waveform each 52 samples recorded with 30kHz rate. Tuple the number of spikes and list of *SegmentDataPacket* classes. E.g.

*fs\_clk=30000*

*spk\_t = np.arange(0, 52000 / fs\_clk, 1000 / fs\_clk, dtype=np.float32)*

*n, seg\_data = xp.spk\_data(elec)*

*for p in seg\_data:*

*plt.plot(spk\_t, p.wf)*

Stim data:

.stim\_data(elec, max\_spk=1024)

Retrieves segment data containing stim waveforms. Tuple the number of events and list of *SegmentDataPacket* classes.

Digin:

.digin(max\_events=1024)

The ‘digin’ function returns events triggered on the Digital I/O Front End. Tuple the number of events and list of *SegmentDataPacket* classes.

Digout:

.digout(outputs, values)

The ‘digout’ function controls digital output ports on the Digital I/O Front End.

*outputs*: list of integers designating desired output channels  
*values*: list of values, all are 0, 1 except the parallel port which is a 16-bit integer.

Close:

.\_close()

Immediately close any open connections to the NIP. \_close should not be called from within a xipppy\_open context.