ephyviewer Documentation

Release 1.6.1.dev0

Samuel Garcia, Jeffrey Gill

Mar 24, 2023

Contents

Simple viewers for ephys signals, events, video and more

Distributions

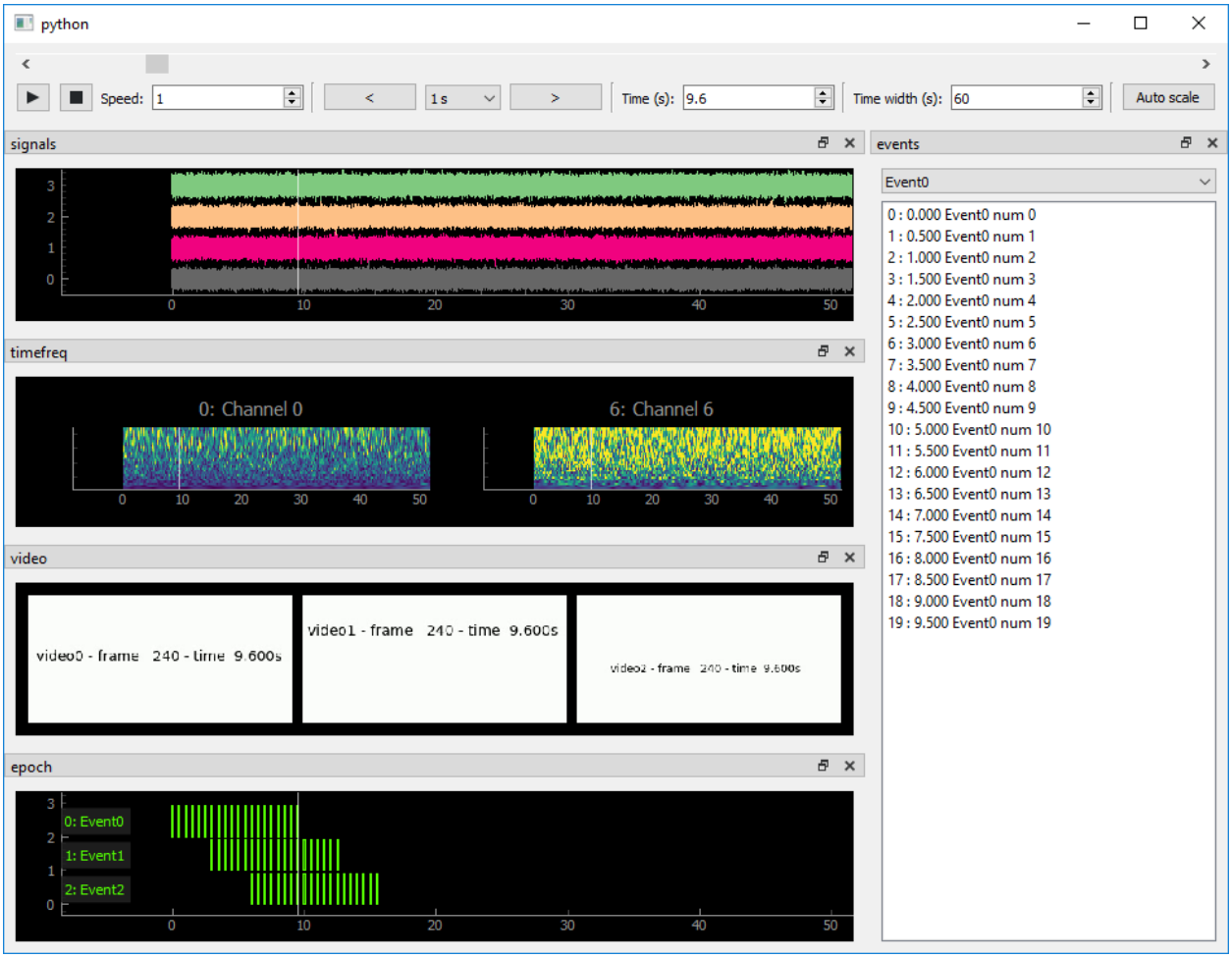
Source Code

Tests Status

Version: 1.6.1.dev0 (other versions)

Do you have a large neural/electrophysiological dataset? Do you want to closely examine the raw signals and other events before performing an in-depth analysis? Good news! **ephyviewer** is your friend.

ephyviewer is both a standalone application and a Python library for creating scripts that fulfill your visualization needs.



For an example of an application that utilizes **ephyviewer**'s capabilities as a library, see the neurotic app and this paper:

> Gill, J. P., Garcia, S., Ting, L. H., Wu, M., & Chiel, H. J. (2020). *neurotic*: Neuroscience Tool for Interactive Characterization. eNeuro, 7(3). https://doi.org/10.1523/ENEURO.0085-20.2020

CHAPTER 1

Table of Contents

1.1 Overview

ephyviewer is a Python library for building custom viewers for electrophysiological signals, video, events, epochs, spike trains, data tables, and time-frequency representations of signals. It also provides an epoch encoder for creating annotations.

ephyviewer can be used as a standalone application (requires Neo 0.6) by launching it from the console, then using the menu to open a data file:

```
ephyviewer
```

See the documentation for `neo.rawio` for available formats.

You can skip the file menu by specifying a filename from the console (and optionally the format, though this can usually be detected automatically):

```
ephyviewer File_axon_1.abf
ephyviewer File_axon_1.abf -f Axon
```

However, where **ephyviewer** really shines is as a library for designing custom viewers in simple Python scripts that meet your individual needs:

```
import ephyviewer
import numpy as np

app = ephyviewer.mkQApp()

# create example signals
sigs = np.random.rand(100000,16)

# create a viewer for the signals
sample_rate = 1000.
t_start = 0.
```

(continues on next page)

(continued from previous page)

```
view1 = ephyviewer.TraceViewer.from_numpy(sigs, sample_rate, t_start, 'Signals')

# create a window
win = ephyviewer.MainViewer()
win.add_view(view1)
win.show()

# launch the app
app.exec()
```

Have a look at the *Examples* to see how to create both simple and sophisticated viewers, and at the *User Interface* guide for how to use the interface.

1.2 Installation

Requirements:

- Python 3.7
- numpy
- scipy
- matplotlib 2.0
- pyqtgraph 0.10.0
- PySide6, PyQt5, PySide2, or PyQt4 (manual installation required)

Optional dependencies:

- Neo 0.6 (standalone app and Neo sources)
- PyAV (video viewer)
- pandas (dataframes and CSV writable epoch sources)
- spikeinterface>=0.92 (for spikeinterface objects)

To install the latest release:

```
pip install ephyviewer
```

To install the latest development version:

```
pip install https://github.com/NeuralEnsemble/ephyviewer/archive/master.zip
```

To install with conda (`python-neo` and `av` are not strictly required but are recommended):

```
conda install -c conda-forge ephyviewer python-neo av
```

1.3 User Interface

The viewers are highly customizable:

- All panels can be hidden, undocked, stacked, or repositioned on the fly.

- Right-click (Control-click on Mac) on the title bar of a visible panel to restore hidden panels.
- Double-click in the plotting region of many viewers to open an options window with additional customization controls.

Time is easy and intuitive to navigate:

- Pressing the play button will scroll through data and video in real time, or at a higher or lower rate if the speed parameter is changed.
- The arrow/WASD keys allow you to step through time in variable increments.
- Dragging the horizontal slider along the top moves through time quickly.
- Jump to a time by clicking on an event in the event list or on an arrow button in the epoch encoder's data table.

It is also easy to quickly adjust scale and placement:

- To show more or less time at once, right-click (Control-click on Mac) and drag right or left to contract or expand time.
- Press the auto scale button to automatically scale signals and color maps.
- Scroll the mouse wheel to zoom in a trace viewer or a video viewer, or to rescale the color map in a time-frequency viewer.
- Scroll the mouse wheel on a trace label to adjust the scale of an individual trace (`by_channel` scale mode only).
- Left-click and drag on a trace label to adjust its vertical offset, or in a video viewer to reposition the video frame.
- Adjust scale mode (real vs arbitrary units) and individual signal gains and offsets in the detailed options window of the trace viewer (double click to open).

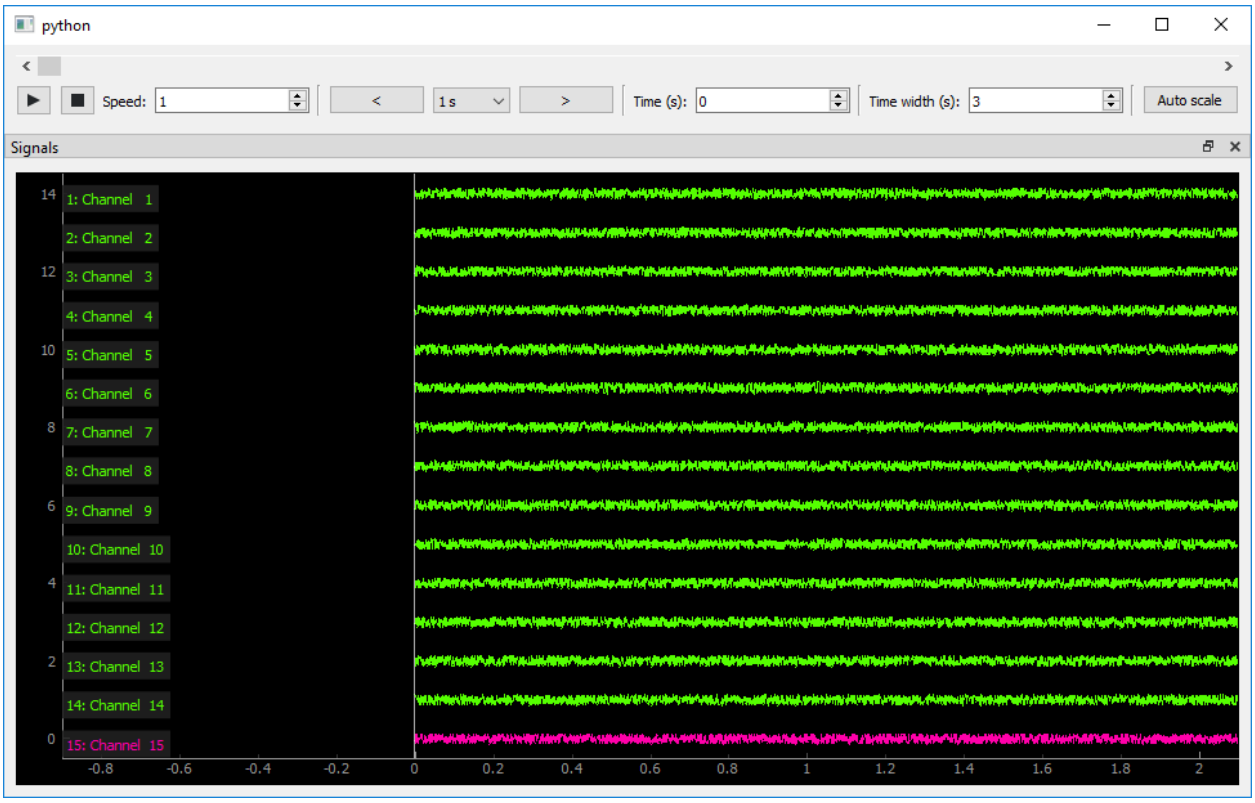
The epoch encoder has many modes of interaction as well:

- Click "Time range selector" (shortcut: `r`) to toggle the display of a rectangle which can be positioned with the mouse or by using dedicated buttons ("Set start >" and "Set stop >"; shortcuts: `[` and `]`).
- Customizable key bindings (default: number keys) allow you to fill the selected time range with an epoch if the selector is enabled, or to place new epochs of a fixed (and adjustable) duration at the current time if it is disabled. Key bindings are displayed to the left of each epoch label.
- Click on an epoch to select the corresponding entry in the epoch encoder's data table, where you can change the times or label associated with it.
- Buttons in each row of the data table allow you to jump to, split, duplicate, or delete the epoch.
- Double-click on an existing epoch to enable the time range selector and position it on that epoch.
- Toggle between epoch insertion modes using the "Allow overlap" button or by modifying shortcut key presses with the Shift key:
  - "Allow overlaps" disabled: Placing an epoch where there is already one will delete the overlapping portion of the old epoch.
  - "Allow overlaps" enabled: Epochs are permitted to overlap.
- Create new possible epoch labels with the "New label" button.
- Merge overlapping and adjacent epochs of the same type with the "Merge neighbors" button.
- Fill gaps between epochs using the "Fill blank" button.
- Click "Save" (shortcut: `Ctrl+s`, or `Cmd+s` on Mac) to write the epochs to a file.

- Click “Undo”/”Redo” (shortcuts: `Ctrl+z`/`Ctrl+y`, or `Cmd+z`/`Cmd+y` on Mac) to move through the history of recent changes made during the current session.

1.4 Examples

1.4.1 Simple signal viewer



`trace_viewer.py`

```
from ephyviewer import mkQApp, MainViewer, TraceViewer
import numpy as np

#you must first create a main Qt application (for event loop)
app = mkQApp()

#create fake 16 signals with 100000 at 10kHz
sigs = np.random.rand(100000,16)
sample_rate = 1000.
t_start = 0.

#Create the main window that can contain several viewers
win = MainViewer(debug=True, show_auto_scale=True)

#create a viewer for signal with TraceViewer
# TraceViewer normally accept a AnalogSignalSource but
# TraceViewer.from_numpy is facitilty function to bypass that
view1 = TraceViewer.from_numpy(sigs, sample_rate, t_start, 'Signals')
```

(continues on next page)

(continued from previous page)

```
#Parameters can be set in script
view1.params['scale_mode'] = 'same_for_all'
view1.params['display_labels'] = True

#And also parameters for each channel
view1.by_channel_params['ch0', 'visible'] = False
view1.by_channel_params['ch15', 'color'] = '#FF00AA'

#This is needed when scale_mode='same_for_all'
#to recompute the gain
#this avoid to push auto_scale button
view1.auto_scale()

#put this veiwer in the main window
win.add_view(view1)

#show main window and run Qapp
win.show()


app.exec()
```

1.4.2 Signal viewer with data source

trace_viewer_datasource.py

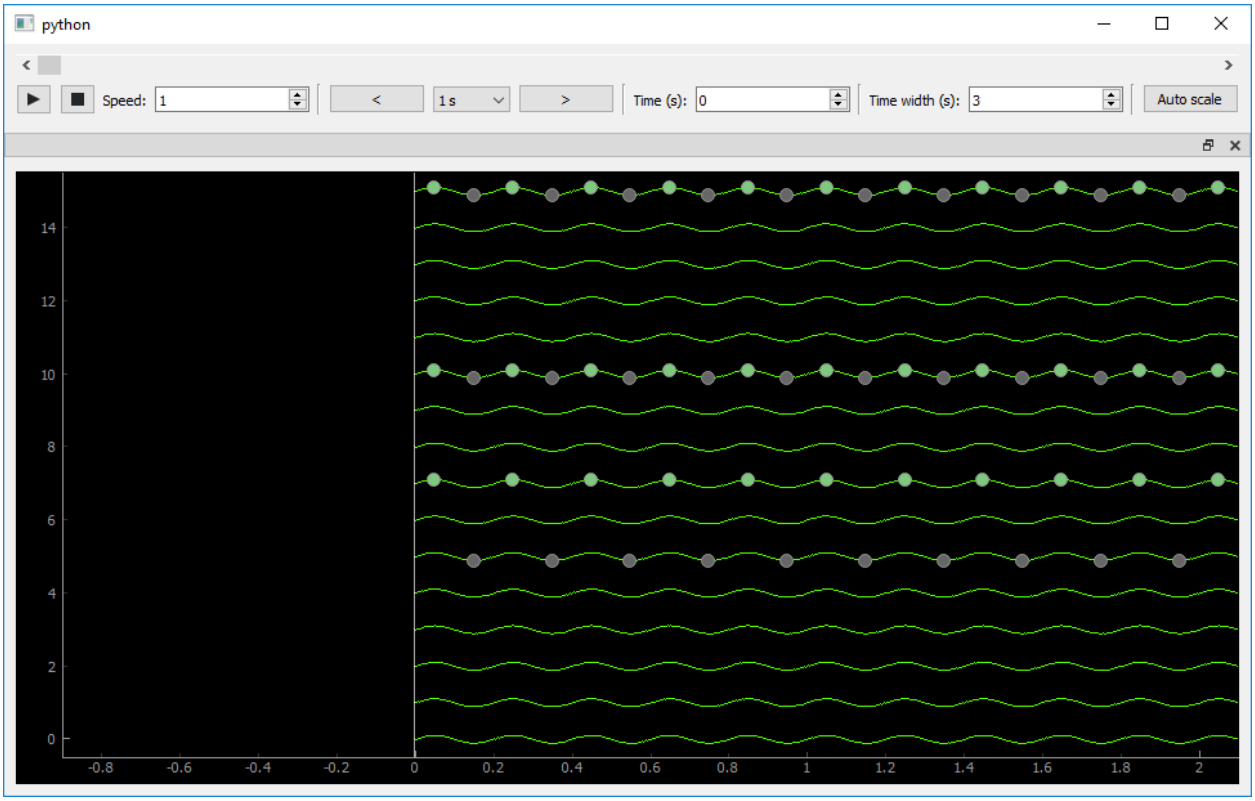
```
from ephyviewer import mkQApp, MainViewer, TraceViewer
from ephyviewer import InMemoryAnalogSignalSource
import ephyviewer
import numpy as np



#you must first create a main Qt application (for event loop)
app = mkQApp()

#create fake 16 signals with 100000 at 10kHz
sigs = np.random.rand(100000,16)
sample_rate = 1000.
t_start = 0.

#Create the main window that can contain several viewers
win = MainViewer(debug=True, show_auto_scale=True)

#Create a datasource for the viewer
# here we use InMemoryAnalogSignalSource but
# you can alose use your custum datasource by inheritance
source = InMemoryAnalogSignalSource(sigs, sample_rate, t_start)

#create a viewer for signal with TraceViewer
# TraceViewer normally accept a AnalogSignalSource but
# TraceViewer.from_numpy is facitilty function to bypass that
view1 = TraceViewer(source=source)
```

(continues on next page)

(continued from previous page)

```
#put this veiwer in the main window
win.add_view(view1)

#show main window and run Qapp
win.show()
app.exec()
```

1.4.3 Signal viewer with markers



trace_viewer_with_marker.py

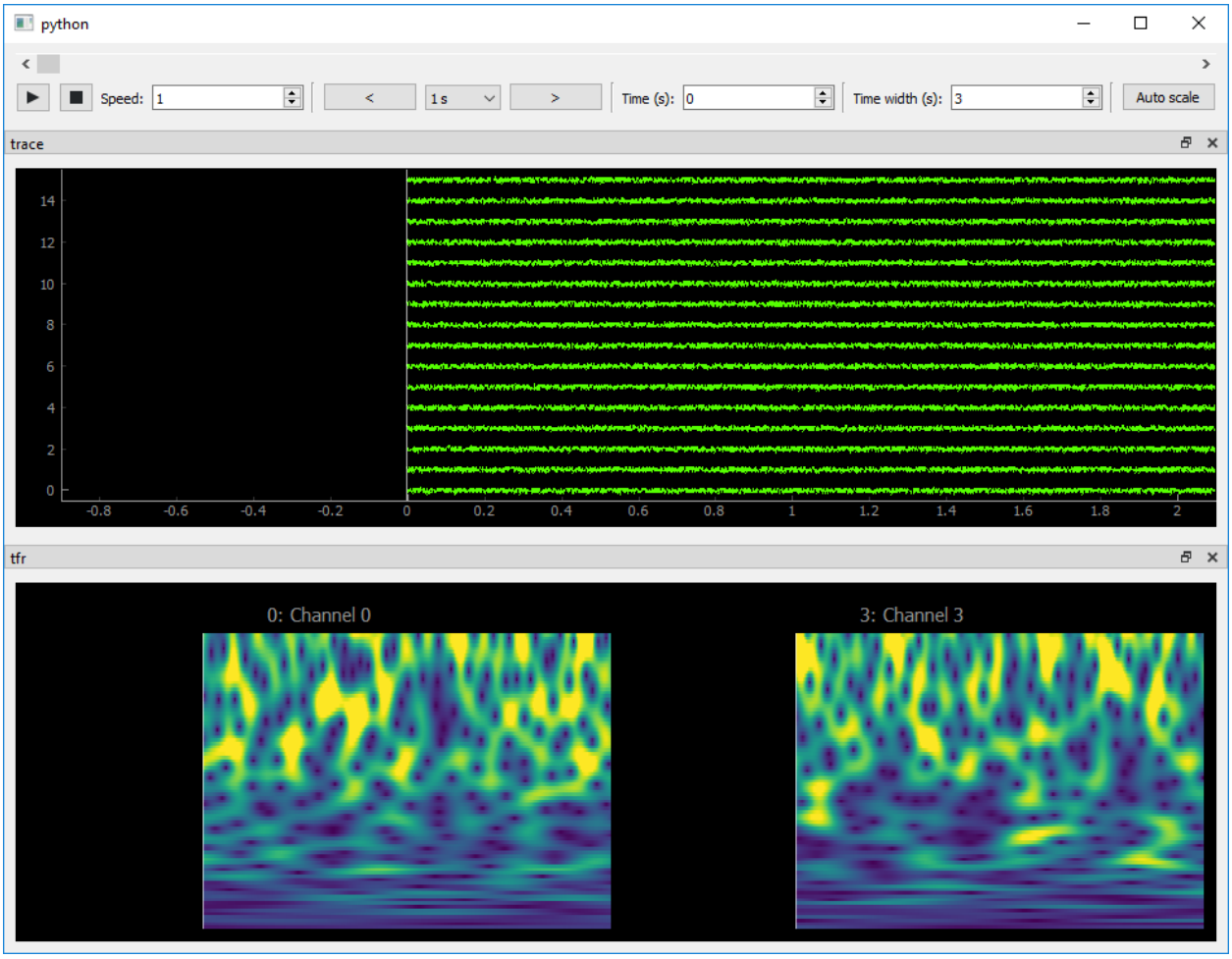
```
from ephyviewer import mkQApp, MainViewer, TraceViewer
from ephyviewer import AnalogSignalSourceWithScatter
import ephyviewer
import numpy as np

#you must first create a main Qt application (for event loop)
app = mkQApp()

#create 16 signals with 100000 at 10kHz
sigs = np.random.rand(100000,16)
sample_rate = 1000.
t_start = 0.


#create fake 16 signals with sinus
```

(continues on next page)

(continued from previous page)

```
sample_rate = 1000.
t_start = 0.
times = np.arange(1000000)/sample_rate
signals = np.sin(times*2*np.pi*5)[:, None]
signals = np.tile(signals, (1, 16))

#detect some crossing zeros
s0 = signals[:-2, 0]
s1 = signals[1:-1,0]
s2 = signals[2:,0]
peaks0,  = np.nonzero((s0<s1) & (s2<s1))
peaks1,  = np.nonzero((s0>s1) & (s2>s1))

#create 2 familly scatters from theses 2 indexes
scatter_indexes = {0: peaks0, 1: peaks1}
#and asign them to some channels each
scatter_channels = {0: [0, 5, 8], 1: [0, 5, 10]}
source = AnalogSignalSourceWithScatter(signals, sample_rate, t_start, scatter_indexes,
↪ scatter_channels)


#Create the main window that can contain several viewers
win = MainViewer(debug=True, show_auto_scale=True)

#create a viewer for signal with TraceViewer
#connected to the signal source
view1 = TraceViewer(source=source)

view1.params['scale_mode'] = 'same_for_all'
view1.auto_scale()

#put this veiwer in the main window
win.add_view(view1)

#show main window and run Qapp
win.show()
app.exec()
```

1.4.4 Time-frequency viewer



timefreq_viewer.py

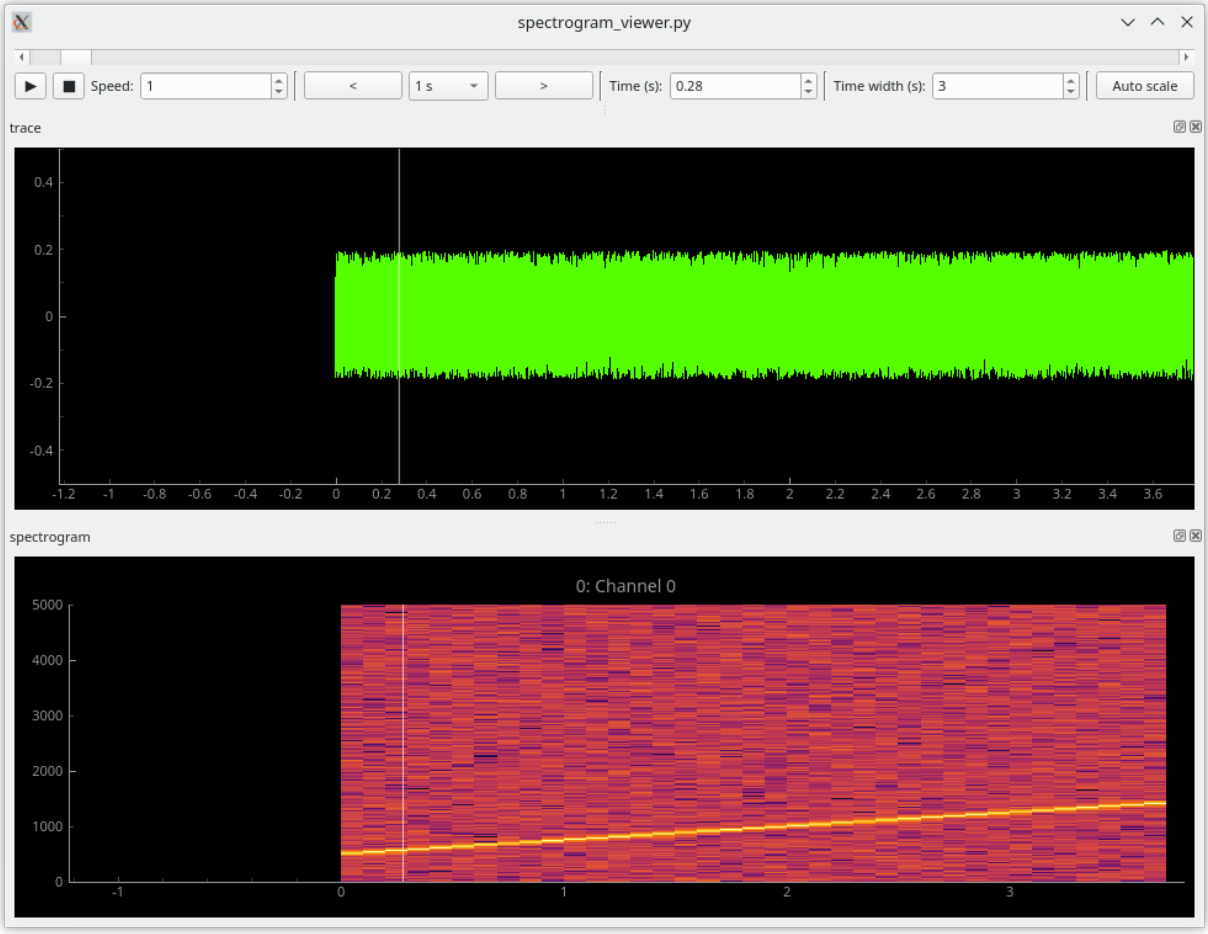
```
from ephyviewer import mkQApp, MainViewer, TraceViewer, TimeFreqViewer
from ephyviewer import InMemoryAnalogSignalSource
import ephyviewer
import numpy as np



#you must first create a main Qt application (for event loop)
app = mkQApp()

#create fake 16 signals with 100000 at 10kHz
sigs = np.random.rand(100000,16)
sample_rate = 1000.
t_start = 0.

#Create the main window that can contain several viewers
win = MainViewer(debug=True, show_auto_scale=True)

#Create a datasource for the viewer
# here we use InMemoryAnalogSignalSource but
```

(continues on next page)

(continued from previous page)

```
# you can alose use your custum datasource by inheritance
source = InMemoryAnalogSignalSource(sigs, sample_rate, t_start)

#create a viewer for signal with TraceViewer
view1 = TraceViewer(source=source, name='trace')
view1.params['scale_mode'] = 'same_for_all'
view1.auto_scale()

#create a time freq viewer conencted to the same source
view2 = TimeFreqViewer(source=source, name='tfr')

view2.params['show_axis'] = False
view2.params['timefreq', 'deltafreq'] = 1
view2.by_channel_params['ch3', 'visible'] = True


#add them to mainwindow
win.add_view(view1)
win.add_view(view2)


#show main window and run Qapp
win.show()
app.exec()
```

1.4.5 Spectrogram viewer



spectrogram_viewer.py

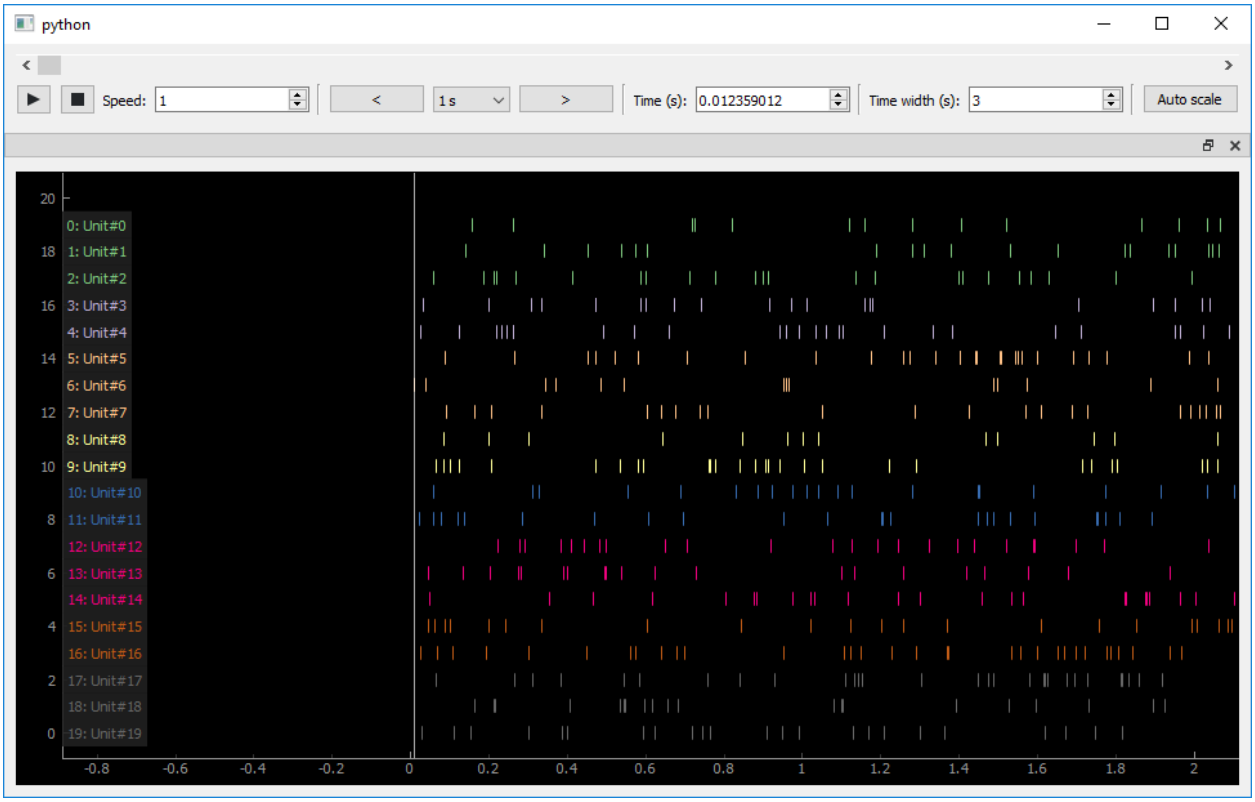
```
from ephyviewer import mkQApp, MainViewer, TraceViewer, SpectrogramViewer
from ephyviewer import InMemoryAnalogSignalSource
import ephyviewer
import numpy as np



# you must first create a main Qt application (for event loop)
app = mkQApp()

# create a fake signals 1 channel at 10kHz
# this emulate a moving frequency
sample_rate = 10000.
duration = 10.
times = np.arange(0, duration, 1./sample_rate, dtype='float64')
sigs = np.random.rand(times.size,1)
t_start = 0.
instantaneous_freqs = np.linspace(500, 3000, times.size)
instantaneous_phase = np.cumsum(instantaneous_freqs / sample_rate)*2*np.pi
sigs[:, 0] += np.sin(instantaneous_phase)
```

(continues on next page)

(continued from previous page)

```
#Create the main window that can contain several viewers
win = MainViewer(debug=True, show_auto_scale=True)

#Create a  signal source for the viewer
source = InMemoryAnalogSignalSource(sigs, sample_rate, t_start)

#create a viewer for signal with TraceViewer
view1 = TraceViewer(source=source, name='trace')
view1.params['scale_mode'] = 'same_for_all'
view1.params['xsize'] = 5.
view1.auto_scale()


#create a SpectrogramViewer on the same source
view2 = SpectrogramViewer(source=source, name='spectrogram')

view2.params['xsize'] = 5.
view2.params['colormap'] = 'inferno'
view2.params['scalogram', 'binsize'] = .1
view2.params['scalogram', 'scale'] = 'dB'
view2.params['scalogram', 'scaling'] = 'spectrum'

#add them to mainwindow
win.add_view(view1)
win.add_view(view2)


#show main window and run Qapp
win.show()
app.exec()
```

1.4.6 Spike train viewer



spikes_viewer.py

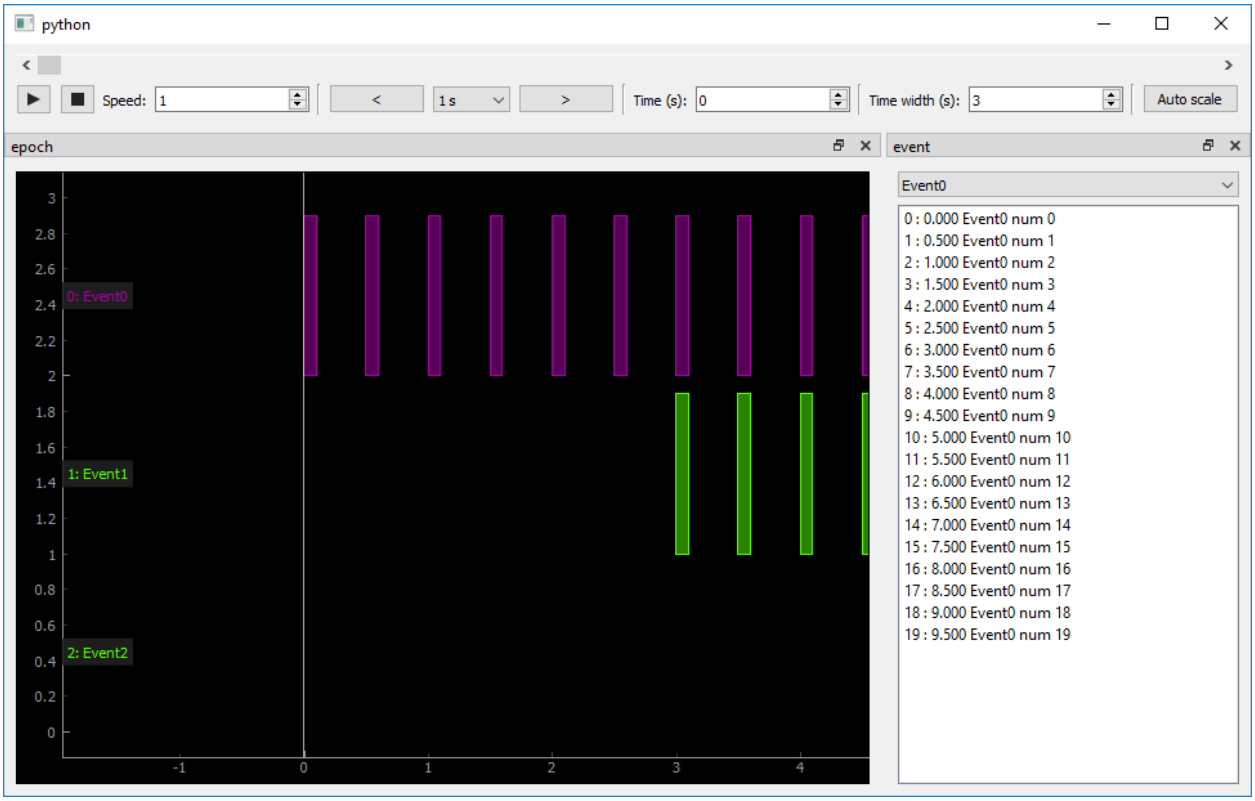
```
from ephyviewer import mkQApp, MainViewer, SpikeTrainViewer
from ephyviewer import InMemorySpikeSource

import numpy as np

#you must first create a main Qt application (for event loop)
app = mkQApp()

#create fake 20 fake units with random firing
#put them in a da ta source
all_spikes =[]
for c in range(20):
    spike_times = np.random.rand(1000)*100.
    spike_times = np.sort(spike_times)
    all_spikes.append({ 'time':spike_times, 'name':'Unit#{}'.format(c) })
source = InMemorySpikeSource(all_spikes=all_spikes)

#Create the main window that can contain several viewers
win = MainViewer(debug=True, show_auto_scale=True)



view1 = SpikeTrainViewer(source=source)


#put this veiwer in the main window
```

(continues on next page)

(continued from previous page)

```
win.add_view(view1)

#show main window and run Qapp
win.show()


app.exec()
```

1.4.7 Epoch viewer



event_epoch_viewer.py

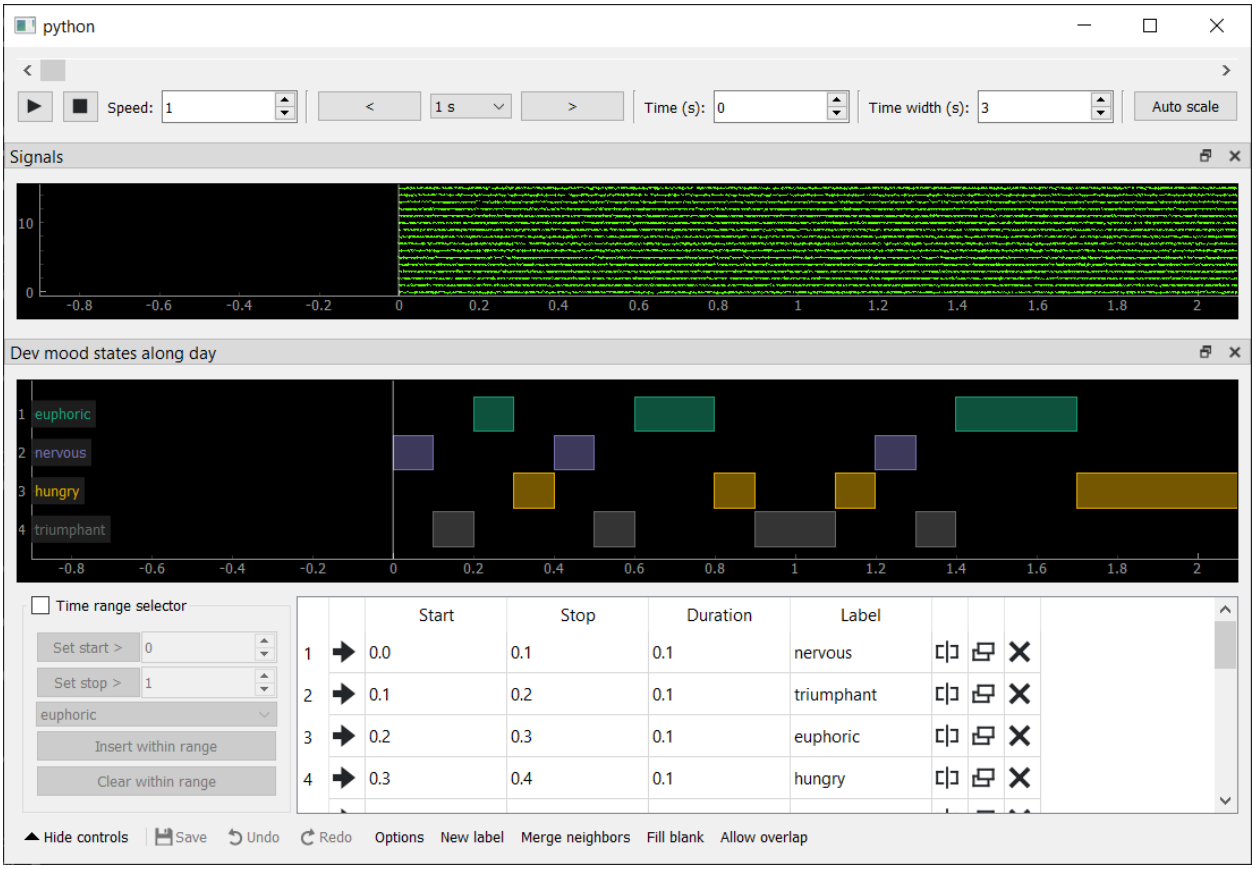
```
from ephyviewer import mkQApp, MainViewer, EpochViewer,EventList
from ephyviewer import InMemoryEventSource, InMemoryEpochSource
import ephyviewer
import numpy as np



#Create one data source with 3 event channel
all_events = []
for c in range(3):
    ev_times = np.arange(0, 10., .5) + c*3
    ev_labels = np.array(['Event{} num {}'.format(c, i) for i in range(ev_times.
↪size)], dtype='U')
    all_events.append({ 'time':ev_times, 'label':ev_labels, 'name':'Event{}'.
↪format(c) })
```

(continues on next page)

(continued from previous page)

```
source_ev = InMemoryEventSource(all_events=all_events)


#Create one data source with 2 epoch channel
all_epochs = []
for c in range(3):
    ep_times = np.arange(0, 10., .5) + c*3
    ep_durations = np.ones(ep_times.shape) * .1
    ep_labels = np.array(['Event{} num {}'.format(c, i) for i in range(ep_times.
↪size)], dtype='U')
    all_epochs.append({ 'time':ep_times, 'duration':ep_durations, 'label':ep_labels,
↪'name':'Event{}'.format(c) })
source_ep = ephyviewer.InMemoryEpochSource(all_epochs=all_epochs)




#you must first create a main Qt application (for event loop)
app = mkQApp()


#Create the main window that can contain several viewers
win = MainViewer(debug=True, show_auto_scale=True)


view1 = EpochViewer(source=source_ep, name='epoch')
view1.by_channel_params['ch0', 'color'] = '#AA00AA'
view1.params['xsize'] = 6.5

view2 = EventList(source=source_ev, name='event')


#add them to mainwindow
win.add_view(view1)
win.add_view(view2, location='bottom',  orientation='horizontal')


#show main window and run Qapp
win.show()
app.exec()
```

1.4.8 Epoch encoder



epoch_encoder.py

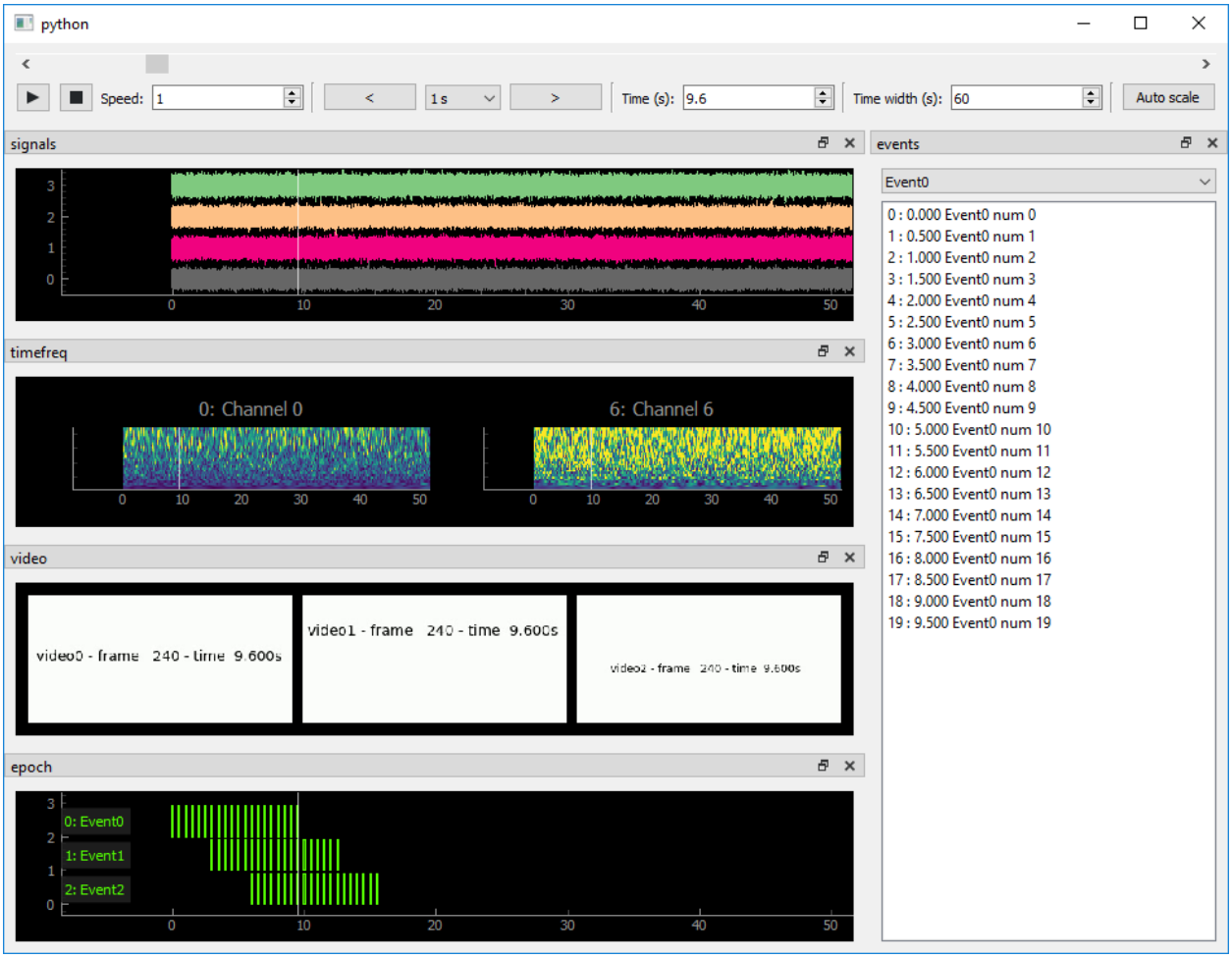
```
"""
ephyviewer also provides an epoch encoder which can be used with shortcut keys
and/or the mouse to encode labels.

ephyviewer makes available a CsvEpochSource class, which inherits from
WritableEpochSource. If you would like to customize reading and writing epochs
to files, you can write your own subclass of WritableEpochSource that implements
the load() and save() methods.

Here is an example of an epoch encoder that uses CsvEpochSource.

"""

from ephyviewer import mkQApp, MainViewer, TraceViewer, CsvEpochSource, EpochEncoder
import numpy as np



# lets encode some dev mood along the day
possible_labels = ['euphoric', 'nervous', 'hungry',  'triumphant']

filename = 'example_dev_mood_encoder.csv'
source_epoch = CsvEpochSource(filename, possible_labels)
```

(continues on next page)

(continued from previous page)

```
#you must first create a main Qt application (for event loop)
app = mkQApp()

#create fake 16 signals with 100000 at 10kHz
sigs = np.random.rand(100000,16)
sample_rate = 1000.
t_start = 0.

#Create the main window that can contain several viewers
win = MainViewer(debug=True, show_auto_scale=True)

#create a viewer for signal
view1 = TraceViewer.from_numpy(sigs, sample_rate, t_start, 'Signals')
view1.params['scale_mode'] = 'same_for_all'
view1.auto_scale()
win.add_view(view1)

#create a viewer for the encoder itself
view2 = EpochEncoder(source=source_epoch, name='Dev mood states along day')
win.add_view(view2)


#show main window and run Qapp
win.show()


app.exec()


# press '1', '2', '3', '4' to encode state.
# or toggle 'Time range selector' and then use 'Insert within range'
```

1.4.9 Mixed viewer



mixed_viewer.py

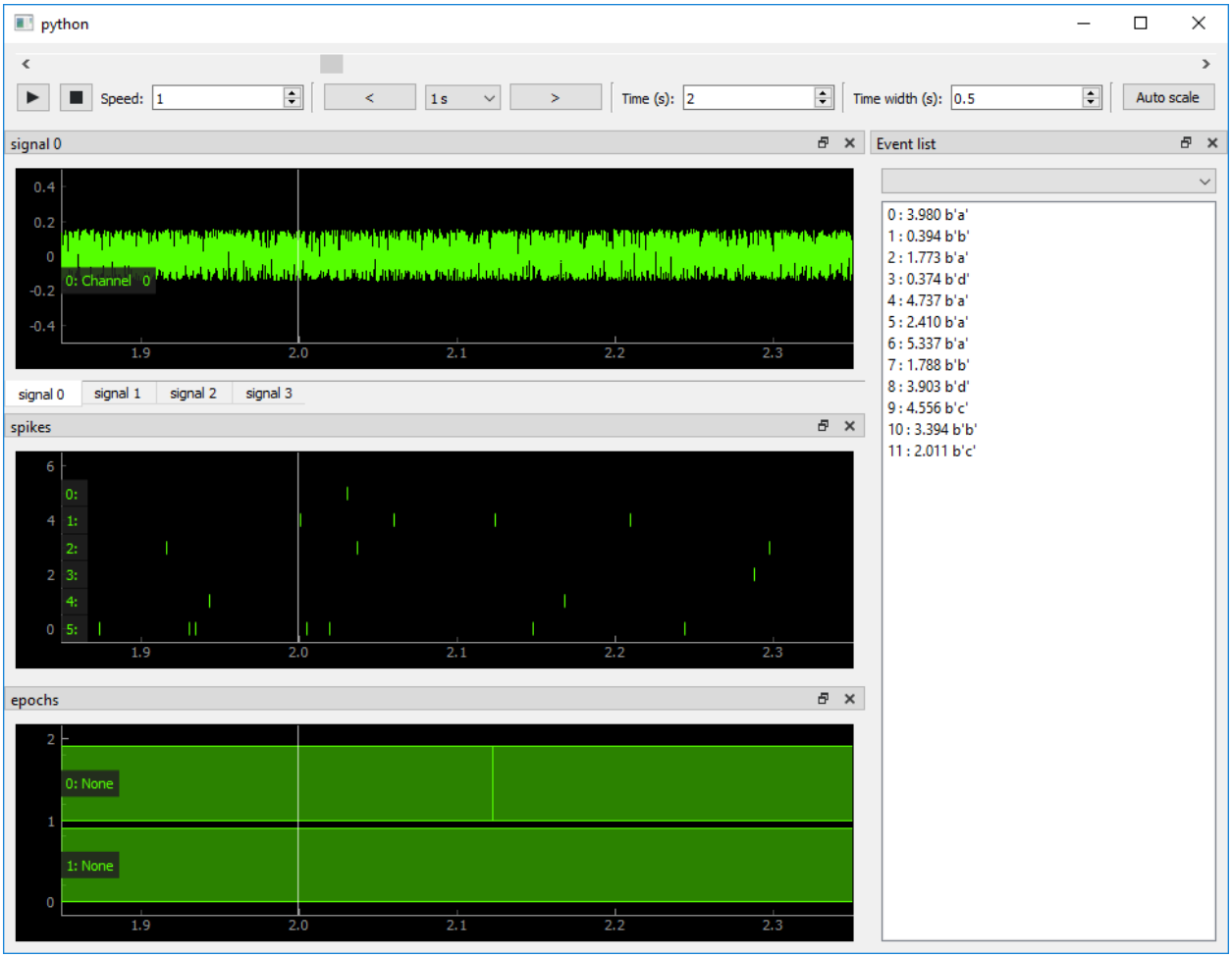
```
import ephyviewer

#for this example we use fake source construct by theses function
from  ephyviewer.tests.testing_tools import make_fake_video_source
from  ephyviewer.tests.testing_tools import make_fake_signals
from  ephyviewer.tests.testing_tools import make_fake_event_source
from  ephyviewer.tests.testing_tools import make_fake_epoch_source

sig_source = make_fake_signals()
event_source = make_fake_event_source()
epoch_source = make_fake_epoch_source()
video_source = make_fake_video_source()



app = ephyviewer.mkQApp()
view1 = ephyviewer.TraceViewer(source=sig_source, name='signals')
view2 = ephyviewer.VideoViewer(source=video_source, name='video')
view3 = ephyviewer.EventList(source=event_source, name='events')
view4 = ephyviewer.EpochViewer(source=epoch_source, name='epoch')
```

(continues on next page)

(continued from previous page)

```
view5 = ephyviewer.TimeFreqViewer(source=sig_source, name='timefreq')


win = ephyviewer.MainViewer(debug=True, settings_name='test1', show_global_xsize=True,
↪ show_auto_scale=True)

win.add_view(view1)
win.add_view(view5, split_with='signals')
win.add_view(view2)
win.add_view(view4)
win.add_view(view3, location='bottom',  orientation='horizontal')

win.show()

win.auto_scale()

app.exec()
```

1.4.10 Viewers from Neo objects



viewers_with_neo_objects.py

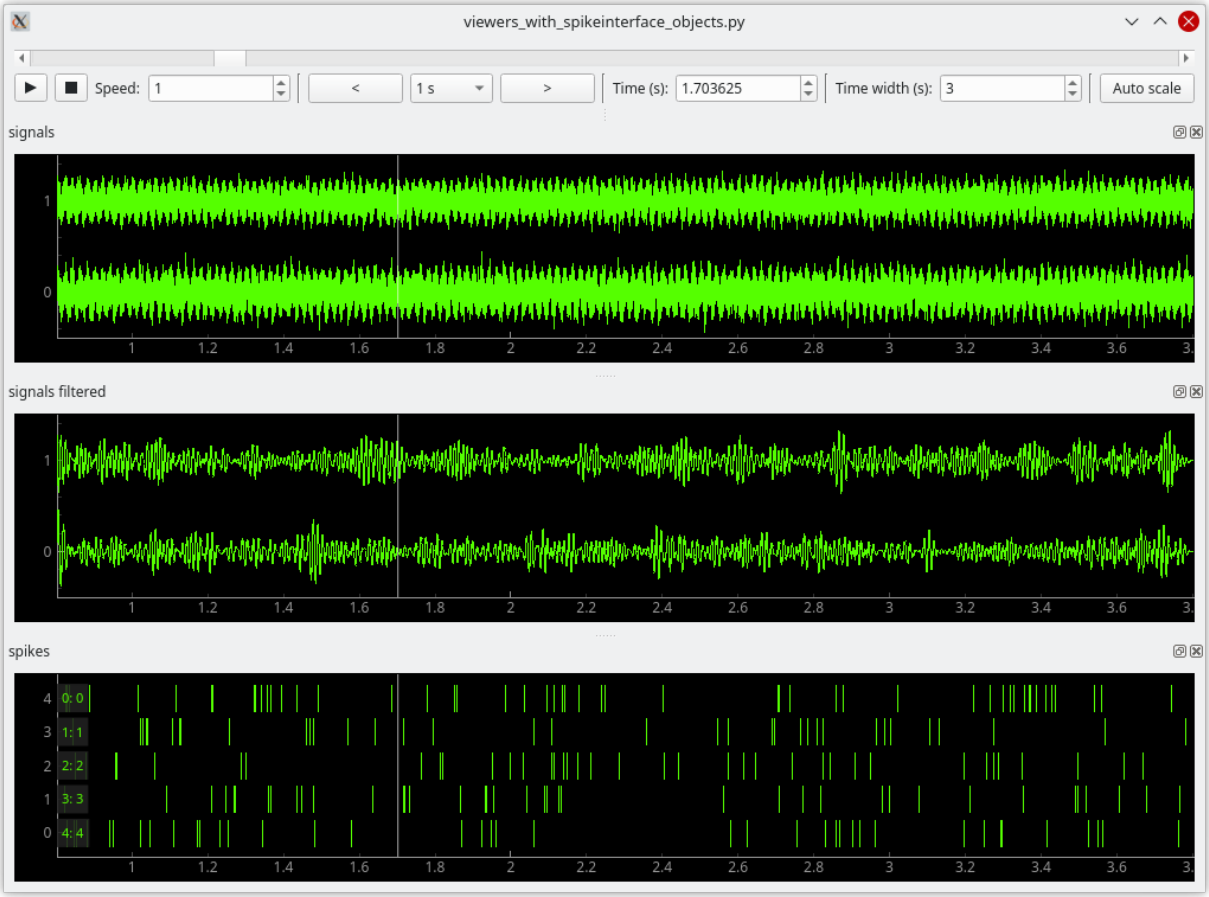
```
"""
Here is an example of opening viewers directly from neo objects.

There are two approaches:
   * create each viewer with class method (TraceViewer.from_neo_analogsignal, ...)
   * magically create all sources by providing the neo.Segment

"""
from ephyviewer import mkQApp, MainViewer, TraceViewer, SpikeTrainViewer, EpochViewer,
↪ EventList
from ephyviewer import get_sources_from_neo_segment, compose_mainviewer_from_sources
import numpy as np


from neo.test.generate_datasets import generate_one_simple_segment
import neo


# here we generate a segment with several objects
# (this is a bad example because it mimics old neo behavior for signals (one␣
↪channel=one object))
neo_seg = generate_one_simple_segment(supported_objects=[neo.Segment, neo.
↪AnalogSignal, neo.Event, neo.Epoch, neo.SpikeTrain])

# the global QT app
app = mkQApp()


#############################
# case 1 : create viewers one at a time directly from neo objects in memory
win = MainViewer(show_auto_scale=True)

# from one neo.AnalogSignal
view1 = TraceViewer.from_neo_analogsignal(neo_seg.analogsignals[0], 'sigs')
win.add_view(view1)

# from several neo.SpikeTrains (3 spiketrains here)
view2 = SpikeTrainViewer.from_neo_spiketrains(neo_seg.spiketrains[0:3], 'spikes')
win.add_view(view2)

# from several neo.Epoch
view3 = EpochViewer.from_neo_epochs(neo_seg.epochs, 'epochs')
win.add_view(view3)

# from several neo.Event
view4 = EventList.from_neo_events(neo_seg.events, 'events')
win.add_view(view4, location='bottom',  orientation='horizontal')

win.show()




#############################
# case 2 : automagically create data sources and a complete window from a neo segment
sources = get_sources_from_neo_segment(neo_seg)
```

(continues on next page)

(continued from previous page)

```
win2 = compose_mainviewer_from_sources(sources)
win2.show()


app.exec()
```

1.4.11 Viewers for SpikeInterface objects



viewers_with_spikeinterface_objects.py

```
"""
Here is an example of opening viewers directly from spikeinterface objects :␣
↪recording and sorting.

In this example, recording and sorting are fake ones generated by spikeinterface but␣
↪of you course
you can use any format supported by spikeinterface.

Note that you can display any lazy preprocessor from
`spikeinterface.toolkit.preprocessing` (filtering, denoising, whittening, ...) so you␣
↪can see immediatly
the clean signal.
```

(continues on next page)

(continued from previous page)

```
"""
import ephyviewer
import spikeinterface.full as si
from spikeinterface.core.testing_tools import generate_recording, generate_sorting


recording = generate_recording()
sig_source = ephyviewer.SpikeInterfaceRecordingSource(recording=recording)

filtered_recording = si.bandpass_filter(recording, freq_min=60., freq_max=100.)
sig_filtered_source = ephyviewer.SpikeInterfaceRecordingSource(recording=filtered_
↪recording)

sorting = generate_sorting()
spike_source = ephyviewer.SpikeInterfaceSortingSource(sorting=sorting)

app = ephyviewer.mkQApp()
win = ephyviewer.MainViewer(debug=True, show_auto_scale=True)

view = ephyviewer.TraceViewer(source=sig_source, name='signals')
win.add_view(view)

view = ephyviewer.TraceViewer(source=sig_filtered_source, name='signals filtered')
win.add_view(view)

view = ephyviewer.SpikeTrainViewer(source=spike_source, name='spikes')
win.add_view(view)


win.show()
app.exec()
```

1.5 Release Notes

1.5.1 Version 1.6.0

2021-03-31

- some fix with pyqtgraph version
- PySide6 integration
- spectrogram implementation
- Prevent accidental un-docking of the navigation toolbar

1.5.2 Version 1.5.1

2021-09-09

Packaging

- Add missing requirements*.txt to source tarball #161

1.5.3 Version 1.5.0

2021-09-09

Compatibility

Support for Python < 3.7 and for PyQt4 were dropped.

- Add compatibility with Neo 0.10.0 (#151, #157, #159)
- Add PySide2 support (#148)

New data sources

- Add data sources for SpikeInterface recording and sorting objects (#153)

Continuous integration

- Run automated test suite with GitHub Actions (#137, #138, #142, #145, #149)
- Add manually-triggerable GitHub Action workflows for publishing to PyPI (#140)
- Add Coveralls test coverage reporting (#144)

1.5.4 Version 1.4.0

2021-01-16

Epoch encoder enhancements

The epoch encoder received major improvements in functionality, appearance, and performance.

- Reorganize EpochEncoder controls to reduce crowding (#113, #108)
- Add editable fields and row buttons to EpochEncoder table (#103, #110, #111, #112, #131)
- Add undo/redo functionality to EpochEncoder (#118, #119, #121)
- Allow new EpochEncoder labels to be created on-the-fly (#105, #127)
- Display EpochEncoder shortcut keys as y-axis tick mark labels (#104)
- Dramatically improve EpochEncoder performance when there are many epochs (#120)

Bug fixes

- Allow standalone app to open multiple files, each in independent windows (#122, #123)
- Fix signal grouping in `get_sources_from_neo_rawio()` for Neo 0.9.0 (#117)
- Fix `'method' object is not connected` TimeFreqViewer error for pyqtgraph 0.11.0 (#124, #126)
- Fix TraceViewer scatter points changing color randomly with pyqtgraph 0.11.1 (#132, #133)
- Fix bounds of EpochEncoder start/stop spin boxes (#106, #107)
- Fix EpochEncoder start/stop spin box values not immediately updating when "Set start"/"Set stop" buttons are pressed on macOS (#115, #129)
- Prevent overzealous spin box height correction in pyqtgraph 0.11.0 (#114)

Documentation

- Update docs style and content (#128, #134, #135)
- Cite eNeuro paper (#125)

1.5.5 Version 1.3.1

2020-03-15

Bug fixes

- Fix TraceViewer auto scale when signals are not in view (#101)

1.5.6 Version 1.3.0

2020-03-01

Enhancements

- Add `xratio` parameter to viewers for controlling vline position (#97)
- Add `label_size` parameter to viewers for resizing channel labels (#98)

1.5.7 Version 1.2.3

2020-02-16

Bug fixes

- Fix plot range for SpikeTrainViewer when only one channel is visible (#95)
- Fix EpochViewer colors when some channels are not visible (#93)

1.5.8 Version 1.2.2

2019-12-18

Bug fixes

- Fix error encountered with some videos: `TypeError: '>=' not supported between instances of 'NoneType' and 'int'` (#91)

1.5.9 Version 1.2.1

2019-12-12

Bug fixes

- Prevent trace labels from moving when zooming (#85)
- Fix plot range (max y) for SpikeTrainViewer (#87)

Enhancements

- Add `auto_scale_factor` param to TraceViewer (#84)
- Add `scatter_size` param to SpikeTrainViewer (#88)

Documentation

- Add conda-forge install instructions to docs (#86)

1.5.10 Version 1.2.0

2019-10-09

Mouse and keyboard enhancements

- Adjust offsets and gains of individual traces in TraceViewer using the mouse (#75)
- Add EpochEncoder shortcuts for toggling range, setting left/right limits (#76)
- Add EpochEncoder shortcut for saving and smarter prompt to save at exit (#77, #78)

Other enhancements

- Add a parameter to NavigationToolBar for setting datetime string format (#70)
- Use scientific notation in TraceViewer gain spin boxes less often (#71)

Documentation

- Add links to RawIO list in Neo's docs (#69)

1.5.11 Version 1.1.1

2019-07-15

Documentation

- Add long description and project URLs to PyPI metadata (#65)

1.5.12 Version 1.1.0

2019-07-15

New major features

- Add functions to make viewers directly from Neo objects (#8, #20)
- Add CsvEpochSource and many EpochEncoder enhancements (#13, #27, #60)
- Add keyboard shortcuts for navigation (#18, #25, #41)
- Add time zoom to most viewers, add global time zoom option (#17)

Aesthetics

- Standardize opacity, color, and z-order for vline marking current time for all viewers (#14)
- Improve label legibility (#19)
- Add `scatter_size` as an adjustable parameter to TraceViewer (#21)
- Remove "s" suffixes from navigation spinboxes for time and xsize (#24)
- Add line-width option to TraceViewer (#26, #43)
- Allow user to change vline color (#45)
- Add param for setting background fill color of channel labels (#49)
- Use same color determination for EpochEncoder rectangles as EpochViewer (#50)

Zoom fixes

- Fix time zoom rate (#23)
- Fix signal offsets when zooming in TraceViewer (#48)

Plot fixes

- Fix time alignment across viewers by removing x-padding (#7)
- Fix poor plotting precision at long times (#28, #29)
- Fix for drawing epochs that end exactly at `t_stop` (#39)
- Fix TraceViewer drawing incomplete curves when zoomed in (#47)
- Appropriately rescale raw signals in AnalogSignalFromNeoRawIOSource (#44, #56, #57)

- Fix trace label and baseline placement for RawIOs with non-zero offset (#58, #61)
- Fix crash when scatter plot channels are hidden (#64)

Video fixes

- VideoViewer performance fixes (#10, #11)
- Fix for video streams lacking `start_time` (#30)
- Address PyAV deprecation warnings (#31)
- Fix for seeking non-existent video frames before start and beyond end of video (#33, #36)
- Ensure `time_to_frame_index` always returns the frame preceding t (#35)
- Clear video image when t is before the start or after the end of video (#37)

Other bug fixes

- Fix missing icons for play and stop buttons (#15, #46)
- Fix errors when dragging or scrolling mouse on viewer axes (#51, #52)
- Fix clicking on current selection in EventList, EpochEncoder table, and DataFrameView (#54, #55)
- Fix for standalone app crash when signals start after t=2.1 (#38)
- Catch `make_video_file` crash (#59)
- 2 bug fixes (#22)
- Some bug fixes (#9)
- Hide debug statements (#32)

Documentation

- Add epoch encoder example (#12)
- Update docs and README (#62)

1.5.13 Version 1.0.1

2018-03-26

- Bug fixes (#5)

1.5.14 Version 1.0.0

2017-09-21

- Initial release