
devicely

Release 1.1.1

Ariane Sasso, Jost Morgenstern, Felix Musmann, Bert Arnrich

Jan 20, 2023

CONTENTS

1	Usage	3
2	About devicely	5
2.1	Start guide for the devicely package	5
2.2	Use devicely	5
2.3	Module reference	22
2.4	How to contribute	29
	Python Module Index	33
	Index	35

The devicely package is made for reading, writing and de-identifying health sensor data from several sensors:

- [Empatica E4](#) is a wearable device that offers real-time physiological data acquisition such as blood volume pulse, electrodermal activity (EDA), heart rate, interbeat intervals, 3-axis acceleration and skin temperature.
- [Biovotion Everion](#) is a wearable device used for the continuous monitoring of vital signs. Currently, it measures the following vital signs: heart rate, blood pulse wave, heart rate variability, activity, SPO2, blood perfusion, respiration rate, steps, energy expenditure, skin temperature, EDA / galvanic skin response (GSR), barometric pressure and sleep.
- [1-lead ECG monitor Faros 180 from Bittium](#) is a one channel ECG monitor with sampling frequency up to 1000 Hz and a 3D acceleration sampling up to 100Hz.
- [Spacelabs \(SL 90217\)](#) is an oscillometric blood pressure (BP) monitor which can be used to automatically track a person's BP in specified time intervals.
- [TimeStamp for Android](#) allows you to record the timestamp of an event at the time it occurs. It also allows you to create specific tags such as "Running" or "Walking" and timestamp those specific activities.
- [Shimmer Consensys GSR](#) is a device that is used to collect sensor data in real time and it contains sensors such as GSR / EDA, photoplethysmography (PPG), 3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer & integrated altimeter.
- [Muse S](#) is a consumer grade headband consisting of 4 electrodes electroencephalography (EEG) sensors, 3-axis accelerometer (ACC), gyroscope, and photoplethysmography (PPG) sensors.

devicely makes it easy to read sensor data and access it in common formats such as dataframes.

We provide a custom reader class for each sensor which has three core methods: read, timeshift and write. The timeshift method changes the time of measurement, thereby automatically helping to de-identify the data.

After timeshifting, you can write the data in the original data format. Now you have one more safety measure in place that will facilitate data sharing.

USAGE

- *Start guide for the devicely package*
- *Module reference*
- Example notebook and data

ABOUT DEVICELY

- [GitHub](#)
- [PyPi](#)
- *How to contribute*

2.1 Start guide for the devicely package

To install devicely locally just run `pip install devicely`

2.2 Use devicely

```
[43]: import os

import devicely

write_dir = 'New'

if not os.path.isdir(write_dir):
    os.makedirs(write_dir)
```

2.2.1 Empatica E4

The Empatice E4 wristband can be used to obtain data from inter-beat intervals, electrodermal activity, heart rate, temperature and blood volume pulse. The wristband uses [this directory structure](#) for its measurement data.

Read the data

Create an EmpaticaReader object:

```
[44]: empatica_reader = devicely.EmpaticaReader('Empatica')
```

Access the sampling frequencies and starting times for all signals:

```
[45]: empatica_reader.start_times
```

```
[45]: {'ACC': Timestamp('2020-10-16 17:04:29'),
      'BVP': Timestamp('2020-10-16 17:04:29'),
      'EDA': Timestamp('2020-10-16 17:04:29'),
      'HR': Timestamp('2020-10-16 17:04:39'),
      'TEMP': Timestamp('2020-10-16 17:04:29'),
      'IBI': Timestamp('2020-10-16 17:04:29')}
```

```
[46]: empatica_reader.sample_freqs
```

```
[46]: {'ACC': 32.0, 'BVP': 64.0, 'EDA': 4.0, 'HR': 1.0, 'TEMP': 4.0}
```

Access the individual dataframes via the attributes ACC, BVP, EDA, HR, TEMP, IBI and tags:

```
[47]: empatica_reader.HR.head()
```

```
[47]: 2020-10-16 17:04:39    51.00
      2020-10-16 17:04:40    51.50
      2020-10-16 17:04:41    51.00
      2020-10-16 17:04:42    52.75
      2020-10-16 17:04:43    64.00
      Name: HR, dtype: float64
```

Access a joined dataframe of all signals:

```
[48]: empatica_reader.data.head()
```

```
[48]:
```

	ACC_X	ACC_Y	ACC_Z	BVP	EDA	HR	TEMP
2020-10-16 17:04:29.000000	-3.0	11.0	60.0	-0.0	0.0	NaN	31.13
2020-10-16 17:04:29.015625	NaN	NaN	NaN	-0.0	NaN	NaN	NaN
2020-10-16 17:04:29.031250	-3.0	11.0	60.0	-0.0	NaN	NaN	NaN
2020-10-16 17:04:29.046875	NaN	NaN	NaN	-0.0	NaN	NaN	NaN
2020-10-16 17:04:29.062500	-2.0	11.0	60.0	-0.0	NaN	NaN	NaN

The dataframe contains nan values because the individual signals have different sampling frequencies.

Timeshift the data

Apply a timeshift:

```
[49]: empatica_reader.timeshift()
      empatica_reader.start_times
```

```
[49]: {'ACC': Timestamp('2019-05-18 22:02:53.817793664'),
      'BVP': Timestamp('2019-05-18 22:02:53.817793664'),
      'EDA': Timestamp('2019-05-18 22:02:53.817793664'),
      'HR': Timestamp('2019-05-18 22:03:03.817793664'),
      'TEMP': Timestamp('2019-05-18 22:02:53.817793664'),
      'IBI': Timestamp('2019-05-18 22:02:53.817793664')}
```

By providing no parameter to `timeshift` the data is shifted by a random time interval between one month and two years to the past. You can also provide a `pandas.Timedelta` object to shift the data by that `timedelta` or a `pandas.Timestamp` object to shift the data such that this timestamp is the earliest entry.

Write the data

```
[50]: empathica_write_path = os.path.join(write_dir, 'Empatica')
      empathica_reader.write(empathica_write_path)
      os.listdir(empathica_write_path)

[50]: ['EDA.csv', 'HR.csv', 'IBI.csv', 'TEMP.csv', 'tags.csv', 'BVP.csv', 'ACC.csv']
```

2.2.2 Spacelabs Blood Pressure Monitor

Spacelabs uses a [single file](#) to output its metadata as well as the actual signals.

Read the data

Create a SpacelabsReader object:

```
[51]: spacelabs_reader = devicely.SpacelabsReader(os.path.join('Spacelabs', 'spacelabs.abp'))
```

Access the metadata:

```
[52]: spacelabs_reader.subject
```

```
[52]: '001V0'
```

```
[53]: spacelabs_reader.metadata
```

```
[53]: {'PATIENTINFO': {'DOB': None, 'RACE': None},
      'REPORTINFO': {'PHYSICIAN': None,
                    'NURSETECH': 'admin',
                    'STATUS': 'NOTCONFIRMED',
                    'CALIPERSUMMARY': {'COUNT': '0'}}}
```

Access the signal dataframe:

```
[54]: spacelabs_reader.data.head()
```

```
[54]:
```

		date	time	SYS(mmHg)	DIA(mmHg)	UNKNOW_1	\
timestamp							
2019-03-01 16:18:00		2019-03-01	16:18:00	107	76	78.0	
2019-03-01 16:19:00		2019-03-01	16:19:00	96	62	63.0	
2019-03-01 16:22:00		2019-03-01	16:22:00	100	68	64.0	
2019-03-01 16:23:00		2019-03-01	16:23:00	103	68	68.0	
2019-03-01 16:25:00		2019-03-01	16:25:00	101	67	65.0	

		UNKNOW_2	UNKNOW_3	CODE
timestamp				
2019-03-01 16:18:00		78.0	NaN	NaN
2019-03-01 16:19:00		63.0	NaN	NaN
2019-03-01 16:22:00		64.0	NaN	NaN
2019-03-01 16:23:00		68.0	NaN	NaN
2019-03-01 16:25:00		65.0	NaN	NaN

Timeshift the data

Apply a timeshift:

```
[55]: spacelabs_reader.timeshift()
      spacelabs_reader.data.head()
```

```
[55]:
```

	date	time	SYS(mmHg)	DIA(mmHg)	UNKNOW_1	\
timestamp						
2018-02-28 18:14:00	2018-02-28	18:14:00	107	76	78.0	
2018-02-28 18:15:00	2018-02-28	18:15:00	96	62	63.0	
2018-02-28 18:18:00	2018-02-28	18:18:00	100	68	64.0	
2018-02-28 18:19:00	2018-02-28	18:19:00	103	68	68.0	
2018-02-28 18:21:00	2018-02-28	18:21:00	101	67	65.0	

	UNKNOW_2	UNKNOW_3	CODE
timestamp			
2018-02-28 18:14:00	78.0	NaN	NaN
2018-02-28 18:15:00	63.0	NaN	NaN
2018-02-28 18:18:00	64.0	NaN	NaN
2018-02-28 18:19:00	68.0	NaN	NaN
2018-02-28 18:21:00	65.0	NaN	NaN

By providing no parameter to `timeshift` the data is shifted by a random time interval between one month and two years to the past. You can also provide a `pandas.Timedelta` object to shift the data by that `timedelta` or a `pandas.Timestamp` object to shift the data such that this timestamp is the earliest entry.

Remove metadata

```
[56]: spacelabs_reader.deidentify('001')
      spacelabs_reader.metadata
```

```
[56]: {'PATIENTINFO': {'DOB': '', 'RACE': ''},
      'REPORTINFO': {'PHYSICIAN': '',
                    'NURSETECH': '',
                    'STATUS': ''},
      'CALIPERSUMMARY': {'COUNT': ''}}
```

Write the data

```
[57]: spacelabs_write_path = os.path.join(write_dir, 'Spacelabs')

      if not os.path.isdir(spacelabs_write_path):
          os.makedirs(spacelabs_write_path)

      spacelabs_reader.write(os.path.join(spacelabs_write_path, 'spacelabs.abp'))
      os.listdir(spacelabs_write_path)

[57]: ['spacelabs.abp']
```

2.2.3 Bittium Faros

The Faros device outputs data in [EDF files](#). These are specifically made for health sensor data and not human-readable.

Read the data

```
[58]: faros_reader = devicely.FarosReader(os.path.join('Faros', 'faros.EDF'))
```

Access metadata:

```
[59]: faros_reader.start_time
```

```
[59]: Timestamp('2019-03-01 16:12:43')
```

```
[60]: faros_reader.sample_freqs
```

```
[60]: {'ECG': 1000.0, 'ACC': 100.0, 'Marker': 1.0, 'HRV': 5.0}
```

```
[61]: faros_reader.units
```

```
[61]: {'ECG': 'uV', 'ACC': 'mg', 'HRV': 'ms'}
```

You can access the individual signals via the ECG, ACC, HRV and Marker attributes:

```
[62]: faros_reader.ACC.head()
```

```
[62]:
```

	X	Y	Z
2019-03-01 16:12:43.000	164.0	23.0	-1172.0
2019-03-01 16:12:43.010	152.0	23.0	-1172.0
2019-03-01 16:12:43.020	152.0	-24.0	-1079.0
2019-03-01 16:12:43.030	117.0	11.0	-985.0
2019-03-01 16:12:43.040	-47.0	246.0	-1125.0

Join the dataframes:

```
[63]: faros_reader.join_dataframes()
faros_reader.data.head()
```

```
[63]:
```

	ECG	ACC_X	ACC_Y	ACC_Z	ACC_mag	Marker	HRV
2019-03-01 16:12:43.000	26.0	164.0	23.0	-1172.0	NaN	0.0	0.0
2019-03-01 16:12:43.001	-6.0	NaN	NaN	NaN	NaN	NaN	NaN
2019-03-01 16:12:43.002	-31.0	NaN	NaN	NaN	NaN	NaN	NaN
2019-03-01 16:12:43.003	-39.0	NaN	NaN	NaN	NaN	NaN	NaN
2019-03-01 16:12:43.004	-17.0	NaN	NaN	NaN	NaN	NaN	NaN

Timeshift the data

Apply a timeshift:

```
[64]: faros_reader.timeshift()
faros_reader.data.head()
```

```
[64]:
```

	ECG	ACC_X	ACC_Y	ACC_Z	ACC_mag	Marker	HRV
2018-02-22 09:32:12.000	26.0	164.0	23.0	-1172.0	NaN	0.0	0.0
2018-02-22 09:32:12.001	-6.0	NaN	NaN	NaN	NaN	NaN	NaN
2018-02-22 09:32:12.002	-31.0	NaN	NaN	NaN	NaN	NaN	NaN
2018-02-22 09:32:12.003	-39.0	NaN	NaN	NaN	NaN	NaN	NaN
2018-02-22 09:32:12.004	-17.0	NaN	NaN	NaN	NaN	NaN	NaN

By providing no parameter to `timeshift` the data is shifted by a random time interval between one month and two years to the past. You can also provide a `pandas.Timedelta` object to shift the data by that `timedelta` or a `pandas.Timestamp` object to shift the data such that this timestamp is the earliest entry.

Write the data

You can write back the data in the original EDF format or to a directory of individual signal files. Writing to a directory is the preferred method. You can find out why this is the case in our [module reference](#).

```
[65]: faros_write_path = os.path.join(write_dir, 'Faros')
      faros_reader.write(faros_write_path)
      os.listdir(faros_write_path)

[65]: ['Marker.csv', 'ECG.csv', 'HRV.csv', 'faros.edf', 'meta.json', 'ACC.csv']
```

You can also create a `FarosReader` from a written directory:

```
[66]: new_faros_reader = devicely.FarosReader(faros_write_path)
      new_faros_reader.ECG.head()

[66]: 2019-03-01 16:12:43.000    26.0
      2019-03-01 16:12:43.001    -6.0
      2019-03-01 16:12:43.002   -31.0
      2019-03-01 16:12:43.003   -39.0
      2019-03-01 16:12:43.004   -17.0
      Freq: L, Name: ECG, dtype: float64
```

You can also save files as EDF if the necessary metadata are still present.

```
[67]: faros_write_new_path = os.path.join(write_dir, 'Faros', 'faros.edf')
      faros_reader.write(faros_write_new_path, file_format='edf')
      os.listdir(faros_write_path)

[67]: ['Marker.csv', 'ECG.csv', 'HRV.csv', 'faros.edf', 'meta.json', 'ACC.csv']
```

2.2.4 Biovotion Everion

The Everion device outputs data in [multiple csv files](#). Each csv file has a `tag` column which specifies the type of measurement. You can see the different tags and what they mean by looking at `EverionReader.SIGNAL_TAGS`, `EverionReader.SENSOR_TAGS` and `EverionReader.FEATURE_TAGS`.

```
[68]: devicely.EverionReader.FEATURE_TAGS

[68]: {14: 'inter_pulse_interval',
      17: 'pis',
      18: 'pid',
```

(continues on next page)

(continued from previous page)

```
77: 'inter_pulse_deviation',
78: 'pis_quality',
79: 'pid_quality'}
```

Read the data

```
[69]: everion_reader = devicely.EverionReader('Everion')
```

If you would like to specify which tags to keep, you can specify this when initializing the reader.

Access the individual dataframes via aggregates, analytics_events, attributes_dailys, everion_events, features, sensors and signals attributes:

```
[70]: everion_reader.signals.head()
```

```
[70]:
```

	count	streamType	tag	time	values	quality
0	806132	2	71	2019-03-01 15:39:58	0.000000	NaN
1	806132	2	13	2019-03-01 15:39:58	21.864220	100.0
2	806132	2	6	2019-03-01 15:39:58	65.000000	85.0
3	806132	2	66	2019-03-01 15:39:58	1.568628	NaN
4	806132	2	12	2019-03-01 15:39:58	18.000000	93.0

Access a joined dataframe of all signals:

```
[71]: everion_reader.data.head()
```

```
[71]:
```

	heart_rate	heart_rate_quality \
time		
2019-03-01 13:23:05.000000000	NaN	NaN
2019-03-01 13:23:05.019607808	NaN	NaN
2019-03-01 13:23:05.039215872	NaN	NaN
2019-03-01 13:23:05.058823680	NaN	NaN
2019-03-01 13:23:05.078431488	NaN	NaN

	oxygen_saturation	oxygen_saturation_quality \
time		
2019-03-01 13:23:05.000000000	NaN	NaN
2019-03-01 13:23:05.019607808	NaN	NaN
2019-03-01 13:23:05.039215872	NaN	NaN
2019-03-01 13:23:05.058823680	NaN	NaN
2019-03-01 13:23:05.078431488	NaN	NaN

	heart_rate_variability \
time	
2019-03-01 13:23:05.000000000	NaN
2019-03-01 13:23:05.019607808	NaN
2019-03-01 13:23:05.039215872	NaN
2019-03-01 13:23:05.058823680	NaN
2019-03-01 13:23:05.078431488	NaN

	heart_rate_variability_quality \
time	

(continues on next page)

(continued from previous page)

2019-03-01 13:23:05.0000000000					NaN
2019-03-01 13:23:05.019607808					NaN
2019-03-01 13:23:05.039215872					NaN
2019-03-01 13:23:05.058823680					NaN
2019-03-01 13:23:05.078431488					NaN
respiration_rate respiration_rate_quality \					
time					
2019-03-01 13:23:05.0000000000		NaN			NaN
2019-03-01 13:23:05.019607808		NaN			NaN
2019-03-01 13:23:05.039215872		NaN			NaN
2019-03-01 13:23:05.058823680		NaN			NaN
2019-03-01 13:23:05.078431488		NaN			NaN
ctemp ctemp_quality ... \					
time					
2019-03-01 13:23:05.0000000000	NaN	NaN	...		
2019-03-01 13:23:05.019607808	NaN	NaN	...		
2019-03-01 13:23:05.039215872	NaN	NaN	...		
2019-03-01 13:23:05.058823680	NaN	NaN	...		
2019-03-01 13:23:05.078431488	NaN	NaN	...		
inter_pulse_interval \					
time					
2019-03-01 13:23:05.0000000000			NaN		
2019-03-01 13:23:05.019607808			NaN		
2019-03-01 13:23:05.039215872			NaN		
2019-03-01 13:23:05.058823680			NaN		
2019-03-01 13:23:05.078431488			NaN		
inter_pulse_interval_deviation led1_data \					
time					
2019-03-01 13:23:05.0000000000			NaN	3028.0	
2019-03-01 13:23:05.019607808			NaN	3309.0	
2019-03-01 13:23:05.039215872			NaN	3454.0	
2019-03-01 13:23:05.058823680			NaN	3207.0	
2019-03-01 13:23:05.078431488			NaN	3079.0	
led2_data led3_data led4_data accx_data \					
time					
2019-03-01 13:23:05.0000000000	2989.0	2924.0	3340.0	368.0	
2019-03-01 13:23:05.019607808	3336.0	3231.0	3417.0	256.0	
2019-03-01 13:23:05.039215872	3443.0	3390.0	3492.0	144.0	
2019-03-01 13:23:05.058823680	3217.0	3126.0	3543.0	96.0	
2019-03-01 13:23:05.078431488	3092.0	2994.0	3538.0	0.0	
accy_data accz_data acc_mag					
time					
2019-03-01 13:23:05.0000000000	2096.0	-3536.0	4126.976617		
2019-03-01 13:23:05.019607808	2016.0	-3808.0	4316.324362		
2019-03-01 13:23:05.039215872	2288.0	-3728.0	4376.489918		
2019-03-01 13:23:05.058823680	2352.0	-3600.0	4301.292829		

(continues on next page)

(continued from previous page)

```
2019-03-01 13:23:05.078431488      2384.0      -3904.0  4574.349353
```

```
[5 rows x 25 columns]
```

Timeshift the data

Apply a timeshift:

```
[72]: everion_reader.timeshift()
everion_reader.data.head()
```

```
[72]:
```

	heart_rate	heart_rate_quality	\
time			
2017-04-01 20:53:51.000000000	NaN		NaN
2017-04-01 20:53:51.019607808	NaN		NaN
2017-04-01 20:53:51.039215872	NaN		NaN
2017-04-01 20:53:51.058823680	NaN		NaN
2017-04-01 20:53:51.078431488	NaN		NaN

	oxygen_saturation	oxygen_saturation_quality	\
time			
2017-04-01 20:53:51.000000000	NaN		NaN
2017-04-01 20:53:51.019607808	NaN		NaN
2017-04-01 20:53:51.039215872	NaN		NaN
2017-04-01 20:53:51.058823680	NaN		NaN
2017-04-01 20:53:51.078431488	NaN		NaN

	heart_rate_variability	\
time		
2017-04-01 20:53:51.000000000	NaN	
2017-04-01 20:53:51.019607808	NaN	
2017-04-01 20:53:51.039215872	NaN	
2017-04-01 20:53:51.058823680	NaN	
2017-04-01 20:53:51.078431488	NaN	

	heart_rate_variability_quality	\
time		
2017-04-01 20:53:51.000000000	NaN	
2017-04-01 20:53:51.019607808	NaN	
2017-04-01 20:53:51.039215872	NaN	
2017-04-01 20:53:51.058823680	NaN	
2017-04-01 20:53:51.078431488	NaN	

	respiration_rate	respiration_rate_quality	\
time			
2017-04-01 20:53:51.000000000	NaN		NaN
2017-04-01 20:53:51.019607808	NaN		NaN
2017-04-01 20:53:51.039215872	NaN		NaN
2017-04-01 20:53:51.058823680	NaN		NaN
2017-04-01 20:53:51.078431488	NaN		NaN

(continues on next page)

(continued from previous page)

		ctemp	ctemp_quality	...	\
time					
2017-04-01	20:53:51.0000000000	NaN	NaN	...	
2017-04-01	20:53:51.019607808	NaN	NaN	...	
2017-04-01	20:53:51.039215872	NaN	NaN	...	
2017-04-01	20:53:51.058823680	NaN	NaN	...	
2017-04-01	20:53:51.078431488	NaN	NaN	...	
		inter_pulse_interval		\	
time					
2017-04-01	20:53:51.0000000000			NaN	
2017-04-01	20:53:51.019607808			NaN	
2017-04-01	20:53:51.039215872			NaN	
2017-04-01	20:53:51.058823680			NaN	
2017-04-01	20:53:51.078431488			NaN	
		inter_pulse_interval_deviation		led1_data	\
time					
2017-04-01	20:53:51.0000000000			NaN	3028.0
2017-04-01	20:53:51.019607808			NaN	3309.0
2017-04-01	20:53:51.039215872			NaN	3454.0
2017-04-01	20:53:51.058823680			NaN	3207.0
2017-04-01	20:53:51.078431488			NaN	3079.0
		led2_data	led3_data	led4_data	accx_data \
time					
2017-04-01	20:53:51.0000000000	2989.0	2924.0	3340.0	368.0
2017-04-01	20:53:51.019607808	3336.0	3231.0	3417.0	256.0
2017-04-01	20:53:51.039215872	3443.0	3390.0	3492.0	144.0
2017-04-01	20:53:51.058823680	3217.0	3126.0	3543.0	96.0
2017-04-01	20:53:51.078431488	3092.0	2994.0	3538.0	0.0
		accy_data	accz_data	acc_mag	
time					
2017-04-01	20:53:51.0000000000	2096.0	-3536.0	4126.976617	
2017-04-01	20:53:51.019607808	2016.0	-3808.0	4316.324362	
2017-04-01	20:53:51.039215872	2288.0	-3728.0	4376.489918	
2017-04-01	20:53:51.058823680	2352.0	-3600.0	4301.292829	
2017-04-01	20:53:51.078431488	2384.0	-3904.0	4574.349353	

[5 rows x 25 columns]

By providing no parameter to `timeshift` the data is shifted by a random time interval between one month and two years to the past. You can also provide a `pandas.Timedelta` object to shift the data by that `timedelta` or a `pandas.Timestamp` object to shift the data such that this timestamp is the earliest entry.

Write the data

Write the data to a directory while keeping the same format as the original. If you used only a subset of tags when initializing the reader, only these tags will be written.

```
[73]: everion_write_path = os.path.join(write_dir, 'Everion')
everion_reader.write(everion_write_path)
os.listdir(everion_write_path)
```

```
[73]: ['sensor_data.csv',
       'analytics_events.csv',
       'aggregates.csv',
       'everion_events.csv',
       'attributes_dailys.csv',
       'signals.csv',
       'features.csv']
```

2.2.5 Shimmer

Shimmer uses a [single CSV file](#), indexed by time of measurement.

Read the data

```
[74]: shimmer_reader = devicely.ShimmerPlusReader(os.path.join('Shimmer', 'shimmer.csv'))
shimmer_reader.data.head()
```

```
[74]: Shimmer_40AC_Timestamp_Unix_CAL  Shimmer_40AC_Accel_LN_X_CAL  \
0      2020-07-28 10:56:50.034      -1.434783
1      2020-07-28 10:56:50.057      -1.402174
2      2020-07-28 10:56:50.074      -1.434783
3      2020-07-28 10:56:50.099      -1.413043
4      2020-07-28 10:56:50.111      -1.445652

      Shimmer_40AC_Accel_LN_Y_CAL  Shimmer_40AC_Accel_LN_Z_CAL  \
0              10.0              0.554348
1              10.0              0.554348
2              10.0              0.554348
3              10.0              0.521739
4              10.0              0.510870

      Shimmer_40AC_Accel_WR_X_CAL  Shimmer_40AC_Accel_WR_Y_CAL  \
0              -3.930580              8.421305
1              -3.923399              8.442849
2              -3.897068              8.428486
3              -3.932974              8.421305
4              -3.944943              8.428486

      Shimmer_40AC_Accel_WR_Z_CAL  Shimmer_40AC_Battery_CAL  \
0              -1.620586              4139.194139
1              -1.599042              4137.728938
2              -1.603830              4111.355311
3              -1.589467              4140.659341
```

(continues on next page)

(continued from previous page)

```

4          -1.661281          4134.798535

Shimmer_40AC_Ext_Exp_A15_CAL Shimmer_40AC_GSR_Range_CAL ... \
0          1684.981685          2.0 ...
1          1673.260073          2.0 ...
2          1901.098901          2.0 ...
3          1722.344322          2.0 ...
4          1678.388278          2.0 ...

Shimmer_40AC_Gyro_X_CAL Shimmer_40AC_Gyro_Y_CAL Shimmer_40AC_Gyro_Z_CAL \
0          0.137405          1.877863          -0.183206
1          0.183206          1.328244          -0.412214
2          0.274809          1.282443          -0.198473
3          0.229008          1.450382          -0.122137
4          0.137405          1.511450          -0.473282

Shimmer_40AC_Int_Exp_A12_CAL Shimmer_40AC_Mag_X_CAL \
0          1680.586081          -0.112444
1          1703.296703          -0.109445
2          1687.179487          -0.107946
3          1650.549451          -0.106447
4          1701.831502          -0.113943

Shimmer_40AC_Mag_Y_CAL Shimmer_40AC_Mag_Z_CAL \
0          -0.916042          -0.047976
1          -0.913043          -0.047976
2          -0.910045          -0.049475
3          -0.901049          -0.050975
4          -0.904048          -0.037481

Shimmer_40AC_Pressure_BMP280_CAL Shimmer_40AC_Temperature_BMP280_CAL \
0          100.435379          33.365878
1          100.429731          33.365878
2          100.441027          33.365878
3          100.441027          33.365878
4          100.438203          33.365878

Shimmer_40AC_Accel_LN_mag
0          10.117604
1          10.113031
2          10.117604
3          10.112809
4          10.116862

```

[5 rows x 22 columns]

Timeshift the data

Apply a timeshift:

```
[75]: shimmer_reader.timeshift()
      shimmer_reader.data.head()
```

```
[75]: Shimmer_40AC_Timestamp_Unix_CAL Shimmer_40AC_Accel_LN_X_CAL \
0      2019-06-20 19:45:55.894      -1.434783
1      2019-06-20 19:45:55.917      -1.402174
2      2019-06-20 19:45:55.934      -1.434783
3      2019-06-20 19:45:55.959      -1.413043
4      2019-06-20 19:45:55.971      -1.445652

      Shimmer_40AC_Accel_LN_Y_CAL Shimmer_40AC_Accel_LN_Z_CAL \
0      10.0      0.554348
1      10.0      0.554348
2      10.0      0.554348
3      10.0      0.521739
4      10.0      0.510870

      Shimmer_40AC_Accel_WR_X_CAL Shimmer_40AC_Accel_WR_Y_CAL \
0      -3.930580      8.421305
1      -3.923399      8.442849
2      -3.897068      8.428486
3      -3.932974      8.421305
4      -3.944943      8.428486

      Shimmer_40AC_Accel_WR_Z_CAL Shimmer_40AC_Battery_CAL \
0      -1.620586      4139.194139
1      -1.599042      4137.728938
2      -1.603830      4111.355311
3      -1.589467      4140.659341
4      -1.661281      4134.798535

      Shimmer_40AC_Ext_Exp_A15_CAL Shimmer_40AC_GSR_Range_CAL ... \
0      1684.981685      2.0 ...
1      1673.260073      2.0 ...
2      1901.098901      2.0 ...
3      1722.344322      2.0 ...
4      1678.388278      2.0 ...

      Shimmer_40AC_Gyro_X_CAL Shimmer_40AC_Gyro_Y_CAL Shimmer_40AC_Gyro_Z_CAL \
0      0.137405      1.877863      -0.183206
1      0.183206      1.328244      -0.412214
2      0.274809      1.282443      -0.198473
3      0.229008      1.450382      -0.122137
4      0.137405      1.511450      -0.473282

      Shimmer_40AC_Int_Exp_A12_CAL Shimmer_40AC_Mag_X_CAL \
0      1680.586081      -0.112444
1      1703.296703      -0.109445
2      1687.179487      -0.107946
3      1650.549451      -0.106447
```

(continues on next page)

(continued from previous page)

```

4          1701.831502          -0.113943

    Shimmer_40AC_Mag_Y_CAL  Shimmer_40AC_Mag_Z_CAL  \
0          -0.916042          -0.047976
1          -0.913043          -0.047976
2          -0.910045          -0.049475
3          -0.901049          -0.050975
4          -0.904048          -0.037481

    Shimmer_40AC_Pressure_BMP280_CAL  Shimmer_40AC_Temperature_BMP280_CAL  \
0          100.435379          33.365878
1          100.429731          33.365878
2          100.441027          33.365878
3          100.441027          33.365878
4          100.438203          33.365878

    Shimmer_40AC_Accel_LN_mag
0          10.117604
1          10.113031
2          10.117604
3          10.112809
4          10.116862

[5 rows x 22 columns]
```

By providing no parameter to `timeshift` the data is shifted by a random time interval between one month and two years to the past. You can also provide a `pandas.Timedelta` object to shift the data by that `timedelta` or a `pandas.Timestamp` object to shift the data such that this timestamp is the earliest entry.

Write the data

```

[76]: shimmer_write_path = os.path.join(write_dir, 'Shimmer')

if not os.path.isdir(shimmer_write_path):
    os.makedirs(shimmer_write_path)

shimmer_reader.write(os.path.join(shimmer_write_path, 'shimmer_write.csv'))
os.listdir(shimmer_write_path)

[76]: ['shimmer_write.csv']
```

2.2.6 Muse

The `devicely.MuseReader` can be used for reading data generated by the Muse S headband.

Read the data

```
[77]: muse_reader = devicely.MuseReader(os.path.join('Muse', 'data.csv'))
muse_reader.data.head()
```

```
[77]:
```

	Delta_TP9	Delta_AF7	Delta_AF8	Delta_TP10	\
TimeStamp					
2021-05-24 20:26:20.103	0.735186	0.752212	0.714283	1.023466	
2021-05-24 20:26:20.588	NaN	NaN	NaN	NaN	
2021-05-24 20:26:21.161	1.063671	0.915956	0.734191	1.023466	
2021-05-24 20:26:22.174	1.226523	0.791329	0.685841	0.550056	
2021-05-24 20:26:23.184	1.009649	0.822014	0.677420	0.801944	

	Theta_TP9	Theta_AF7	Theta_AF8	Theta_TP10	\
TimeStamp					
2021-05-24 20:26:20.103	0.251417	0.205159	0.208760	0.428435	
2021-05-24 20:26:20.588	NaN	NaN	NaN	NaN	
2021-05-24 20:26:21.161	1.120400	0.367509	0.303293	0.428435	
2021-05-24 20:26:22.174	1.289343	0.647703	0.187995	0.070321	
2021-05-24 20:26:23.184	1.035277	0.500738	-0.049515	0.673528	

	Alpha_TP9	Alpha_AF7	...	Gyro_X	Gyro_Y	\
TimeStamp			...			
2021-05-24 20:26:20.103	0.493829	0.269544	...	3.416901	0.605621	
2021-05-24 20:26:20.588	NaN	NaN	...	NaN	NaN	
2021-05-24 20:26:21.161	0.974990	0.348580	...	4.643097	2.355194	
2021-05-24 20:26:22.174	0.726045	0.420135	...	4.239349	0.635529	
2021-05-24 20:26:23.184	0.805443	0.446498	...	4.366455	-0.052338	

	Gyro_Z	HeadBandOn	HSI_TP9	HSI_AF7	HSI_AF8	\
TimeStamp						
2021-05-24 20:26:20.103	3.768311	1.0	2.0	1.0	1.0	
2021-05-24 20:26:20.588	NaN	NaN	NaN	NaN	NaN	
2021-05-24 20:26:21.161	2.026215	1.0	1.0	1.0	1.0	
2021-05-24 20:26:22.174	3.484192	1.0	1.0	1.0	1.0	
2021-05-24 20:26:23.184	1.525269	1.0	1.0	1.0	1.0	

	HSI_TP10	Battery	Elements
TimeStamp			
2021-05-24 20:26:20.103	2.0	100.0	NaN
2021-05-24 20:26:20.588	NaN	NaN	/muse/elements/blink
2021-05-24 20:26:21.161	1.0	100.0	NaN
2021-05-24 20:26:22.174	1.0	100.0	NaN
2021-05-24 20:26:23.184	1.0	100.0	NaN

[5 rows x 38 columns]

Timeshift the data

Apply a random timeshift:

```
[78]: muse_reader.timeshift()
muse_reader.data.head()
```

```
[78]:
```

	Delta_TP9	Delta_AF7	Delta_AF8	Delta_TP10	\
TimeStamp					
2020-05-30 05:03:58.183587364	0.735186	0.752212	0.714283	1.023466	
2020-05-30 05:03:58.668587364	NaN	NaN	NaN	NaN	
2020-05-30 05:03:59.241587364	1.063671	0.915956	0.734191	1.023466	
2020-05-30 05:04:00.254587364	1.226523	0.791329	0.685841	0.550056	
2020-05-30 05:04:01.264587364	1.009649	0.822014	0.677420	0.801944	

	Theta_TP9	Theta_AF7	Theta_AF8	Theta_TP10	\
TimeStamp					
2020-05-30 05:03:58.183587364	0.251417	0.205159	0.208760	0.428435	
2020-05-30 05:03:58.668587364	NaN	NaN	NaN	NaN	
2020-05-30 05:03:59.241587364	1.120400	0.367509	0.303293	0.428435	
2020-05-30 05:04:00.254587364	1.289343	0.647703	0.187995	0.070321	
2020-05-30 05:04:01.264587364	1.035277	0.500738	-0.049515	0.673528	

	Alpha_TP9	Alpha_AF7	...	Gyro_X	Gyro_Y	\
TimeStamp			...			
2020-05-30 05:03:58.183587364	0.493829	0.269544	...	3.416901	0.605621	
2020-05-30 05:03:58.668587364	NaN	NaN	...	NaN	NaN	
2020-05-30 05:03:59.241587364	0.974990	0.348580	...	4.643097	2.355194	
2020-05-30 05:04:00.254587364	0.726045	0.420135	...	4.239349	0.635529	
2020-05-30 05:04:01.264587364	0.805443	0.446498	...	4.366455	-0.052338	

	Gyro_Z	HeadBandOn	HSI_TP9	HSI_AF7	\
TimeStamp					
2020-05-30 05:03:58.183587364	3.768311	1.0	2.0	1.0	
2020-05-30 05:03:58.668587364	NaN	NaN	NaN	NaN	
2020-05-30 05:03:59.241587364	2.026215	1.0	1.0	1.0	
2020-05-30 05:04:00.254587364	3.484192	1.0	1.0	1.0	
2020-05-30 05:04:01.264587364	1.525269	1.0	1.0	1.0	

	HSI_AF8	HSI_TP10	Battery	\
TimeStamp				
2020-05-30 05:03:58.183587364	1.0	2.0	100.0	
2020-05-30 05:03:58.668587364	NaN	NaN	NaN	
2020-05-30 05:03:59.241587364	1.0	1.0	100.0	
2020-05-30 05:04:00.254587364	1.0	1.0	100.0	
2020-05-30 05:04:01.264587364	1.0	1.0	100.0	

	Elements
TimeStamp	
2020-05-30 05:03:58.183587364	NaN
2020-05-30 05:03:58.668587364	/muse/elements/blink
2020-05-30 05:03:59.241587364	NaN
2020-05-30 05:04:00.254587364	NaN
2020-05-30 05:04:01.264587364	NaN

(continues on next page)

(continued from previous page)

[5 rows x 38 columns]

Write the data

```
[79]: muse_write_path = os.path.join(write_dir, 'muse_write_data.csv')
```

```
if os.path.isfile(muse_write_path):
    os.remove(muse_write_path)
```

```
muse_reader.write(muse_write_path)
os.listdir(write_dir)
```

```
[79]: ['Spacelabs',
      'Everion',
      'muse_write_data.csv',
      'Empatica',
      'Faros',
      'Tags',
      'Shimmer']
```

2.2.7 Tags

You can use the `TimeStampReader` to read data created by the Android app `TimeStamp`. Researchers use this app to mark important times during experiments. The format simple, as can be seen in this [example file](#).

Read the data

```
[80]: timestamp_reader = devicely.TimeStampReader(os.path.join('Tags', 'tags.csv'))
timestamp_reader.data.head()
```

```
[80]:
```

	time	tag_number	tag
	2019-03-01 16:16:37	1	Shake
	2019-03-01 16:17:43	2	Start
	2019-03-01 16:18:20	3	BP Measurement
	2019-03-01 16:19:51	4	BP Measurement
	2019-03-01 16:22:00	5	BP Measurement

Timeshift the data

Apply a timeshift:

```
[81]: timestamp_reader.timeshift()
timestamp_reader.data.head()
```

```
[81]:
```

	time	tag_number	tag
--	------	------------	-----

(continues on next page)

(continued from previous page)

```

2017-09-12 21:26:48      1      Shake
2017-09-12 21:27:54      2      Start
2017-09-12 21:28:31      3 BP Measurement
2017-09-12 21:30:02      4 BP Measurement
2017-09-12 21:32:11      5 BP Measurement

```

By providing no parameter to `timeshift` the data is shifted by a random time interval between one month and two years to the past. You can also provide a `pandas.Timedelta` object to shift the data by that `timedelta` or a `pandas.Timestamp` object to shift the data such that this timestamp is the earliest entry.

Write the data

```

[82]: tags_write_path = os.path.join(write_dir, 'Tags')

if not os.path.isdir(tags_write_path):
    os.makedirs(tags_write_path)

timestamp_reader.write(os.path.join(tags_write_path, 'tags_write.csv'))
os.listdir(tags_write_path)

[82]: ['tags_write.csv']

```

2.3 Module reference

2.3.1 `devicely.EmpaticaReader`

Empatica E4 is a wearable device that offers real-time physiological data acquisition such as blood volume pulse, electrodermal activity (EDA), heart rate, interbeat intervals, 3-axis acceleration and skin temperature.

class `devicely.empatica.EmpaticaReader`(*path*)

Read, timeshift and write data generated by Empatica E4.

start_times

Contain the timestamp of the first measurement for all measured signals (BVP, ACC, etc.).

Type

dict

sample_freqs

Contain the sampling frequencies of all measured signals in Hz.

Type

dict]

IBI

Contain inter-beat interval data. The column “seconds_since_start” is the time in seconds between the start of measurements and the column “IBI” is the duration in seconds between consecutive beats.

Type

pandas.DataFrame

ACC

Contain the data measured with the onboard MEMS type 3-axis accelerometer, indexed by time of measurement.

Type

pandas.DataFrame

BVP

Contain blood volume pulse data, indexed by time of measurement.

Type

pandas.DataFrame

EDA

Contain data captured from the electrodermal activity sensor, indexed by time of measurement.

Type

pandas.DataFrame

HR

Contain heart rate data, indexed by time of measurement.

Type

pandas.DataFrame

TEMP

Contain temperature data, indexed by time of measurement.

Type

pandas.DataFrame

data

Joined dataframe of the ACC, BVP, EDA, HR and TEMP dataframes (see above). May contain NaN values because sampling frequencies differ across signals.

Type

pandas.DataFrame

timeshift(*shift='random'*)

Timeshift all time related columns as well as the starting_times dict.

Parameters

shift (*None*/*'random'*, *pd.Timestamp* or *pd.Timedelta*) – If shift is not specified, shifts the data by a random time interval between one month and two years to the past.

If shift is a timedelta, adds that timedelta to all time-related attributes.

If shift is a timestamp, shifts the data such that the earliest entry has that timestamp. The remaining values will maintain the same time difference to the first entry.

write(*dir_path*)

Write the signal dataframes back to individual csv files formatted the same way as they were read.

Parameters

path (*str*) – Path of the directory in which the csv files are created.

If the directory exists, the csv files are written using writing mode 'w' ignoring other files in the directory.

If the directory does not exist, it will be created.

2.3.2 devicely.SpacelabsReader

Spacelabs (SL 90217) is an oscillometric blood pressure (BP) monitor which can be used to automatically track a person's BP in specified time intervals.

class `devicely.spacelabs.SpacelabsReader`(*path*)

Read, timeshift, deidentify and write data generated by Spacelabs(SL90217).

data

DataFrame with the values that were read from the abp file.

Type

DataFrame

subject

Contain the subject's id. Can be changed for deidentification.

Type

str

valid_measurements

Contain the number of valid measurements in the abp file.

Type

str

metadata

The measurements' metadata. Read from the xml at the bottom of the abp file. Can be erased for deidentification.

Type

dict

deidentify(*subject_id=None*)

Deidentify the data by removing the original XML metadata and subject id.

Parameters

subject_id (*str*, *optional*) – New subject id to be written in the deidentified file, by default None.

timeshift(*shift='random'*)

Timeshift the data by shifting all time related columns.

Parameters

shift (*None*/*'random'*, *pd.Timestamp* or *pd.Timedelta*) – If shift is not specified, shifts the data by a random time interval between one month and two years to the past.

If shift is a timedelta, shifts the data by that timedelta.

If shift is a timestamp, shifts the data such that the earliest entry has that timestamp. The remaining values will maintain the same time difference to the first entry.

write(*path*)

Write the signals and metadata to the writing path in the same format as it was read.

Parameters

path (*str*) – Path to writing file. Writing mode: 'w'. Use the file extension 'abp' to keep the SpaceLabs standard.

2.3.3 devicely.FarosReader

1-lead ECG monitor Faros™ 180 from Bittium is a one channel ECG monitor with sampling frequency up to 1000 Hz and a 3D acceleration sampling up to 100Hz.

class devicely.faros.FarosReader(*path*)

Read, timeshift and write data generated by Bittium Faros devices.

start_time

Start time of all measurements.

Type

pandas.Timestamp

sample_freqs

Sampling frequencies of all signals in Hz.

Type

dict

units

Units of all signals

Type

dict

ECG

ECG signal, indexed by timestamp.

Type

pandas.Series

ACC

Three ACC axes, indexed by timestamp.

Type

pandas.DataFrame

Marker

Markers, indexed by timestamp.

Type

pandas.Series

HRV

HRV signal, indexed by timestamp.

Type

pandas.Series

data

Contain all signals (ECG, ACC, Marker, HRV) indexed by timestamp. Since the signals have different sampling frequencies, many values will be NaN.

Type

DataFrame

join_dataframes()

Join the individual signal dataframes by timestamp. The resulting dataframe is saved in the attribute reader.data.

timeshift(*shift='random'*)

Timeshift the data by shifting all time related values (i.e. `start_time` and `data.index`).

Parameters

shift (*None*/*'random'*, *pd.Timestamp* or *pd.Timedelta*) – If shift is not specified, shifts the data by a random time interval between one month and two years to the past.

If shift is a `timedelta`, shifts the data by that `timedelta`.

If shift is a `timestamp`, shifts the data such that the earliest entry has that timestamp. The remaining values will maintain the same time difference to the first entry.

write(*path, file_format='directory'*)

Write the data either to an EDF file or to several files into a new directory. Because of the [special structure of EDF files](#) writing to EDF is only possible for readers that have been created from an EDF file and without any changes to the ACC, ECG, Marker, HRV and `sample_freqs` attributes. Because we want you to be able to modify the signals, you can write the data back to a directory of individual files. Writing to a directory is the preferred method and works in all cases.

Parameters

- **path** (*str*) – Name of the file or directory to write the data to.
- **file_format** (*{'directory', 'edf', default 'directory'}*) – Format of the written data.

2.3.4 devicely.EverionReader

” Biovotion Everion is a wearable device used for the continuous monitoring of vital signs. Currently, it measures the following vital signs: heart rate, blood pulse wave, heart rate variability, activity, SPO2, blood perfusion, respiration rate, steps, energy expenditure, skin temperature, EDA / galvanic skin response (GSR), barometric pressure and sleep.

class `devicely.everion.EverionReader`(*path, signal_tags=[6, 7, 11, 12, 15, 19, 20, 21, 118, 119], sensor_tags=[80, 81, 82, 83, 84, 85, 86], feature_tags=[14])*

Read, timeshift and write data generated by Biovotion Everion. You can find example data [here](#). Each measurement has a tag which is an information about the type of measurement. By default, only some of the tags are read for signals, sensors and features. You can find these tags via `self.default_signal_tags`, `self.default_sensor_tags` and `self.default_feature_tags`. You can find out all available tags and what they mean by accessing the attributes `EverionReader.SIGNAL_TAGS`, `EverionReader.SENSOR_TAGS` and `EverionReader.FEATURE_TAGS`.

raw dataframes

These dataframes are accessible via the attributes `aggregates`, `analytics_events`, `attributes_dailys`, `everion_events`, `features`, `sensors` and `signals`. Each dataframe corresponds to a file in the reading path.

Type

`dataframes`

data

Joined version of the raw dataframes (`aggregates`, `analytics_events`, `attributes_dailys`, `everion_events`, `features`, `sensors` and `signals`).

Type

`DataFrame`

SIGNAL_TAGS

Signal tag numbers and their meaning.

Type
dict

SENSOR_TAGS

Sensor tag numbers and their meaning.

Type
dict

FEATURE_TAGS

Feature tag numbers and their meaning.

Type
dict

default_signal_tags

Subset of tags that are read by default for signals.

Type
list

default_sensor_tags

Subset of tags that are read by default for sensors.

Type
list

default_feature_tags

Subset of tags that are read by default for features.

Type
list

timeshift(*shift*='random')

Timeshift the data by shifting all time related columns in the joined dataframe (data) and raw dataframes (aggregates, analytics_events, attributes_dailys, everion_events, features, sensors, signals).

Parameters

shift (*None*/'random', *pd.Timestamp* or *pd.Timedelta*) – If shift is not specified, shifts the data by a random time interval between one month and two years to the past.

If shift is a timedelta, shifts the data by that timedelta.

If shift is a timestamp, shifts the data such that the earliest entry has that timestamp. The remaining values will maintain the same time difference to the first entry.

write(*path*)

Write the raw dataframes back to individual files in the same format as they were read. Thus, the resulting directory can be read again using an EverionReader.

Parameters

path (*str*) – Path to the writing directory.

2.3.5 devicely.ShimmerReader

Shimmer Consensys GSR is a device that is used to collect sensor data in real time and it contains sensors such as GSR / EDA, photoplethysmography (PPG), 3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer & integrated altimeter.

class devicely.shimmer_plus.ShimmerPlusReader(*path*)

Read, timeshift and write data generated by Shimmer Consensys GSR.

data

DataFrame with the read signals. The column 'Shimmer_40AC_Timestamp_Unix_CAL' contains the timestamps of measurement.

Type

DataFrame

units

A unit for each column in the dataframe.

Type

pandas.Series

timeshift(*shift='random'*)

Timeshift the data by shifting all time related columns.

Parameters

shift (*None*/'random', *pd.Timestamp* or *pd.Timedelta*) – If shift is not specified, shifts the data by a random time interval between one month and two years to the past.

If shift is a timedelta, shifts the data by that timedelta.

If shift is a timestamp, shifts the data such that the earliest entry has that timestamp. The remaining values will maintain the same time difference to the first entry.

write(*path*)

Write the DataFrame and units to the writing path in the same format as it was read.

Parameters

path (*str*) – Path to writing csv file. Writing mode: 'w'.

2.3.6 devicely.MuseReader

Module to process Muse data from the mind monitor application

class devicely.muse.MuseReader(*path*)

Parses, timeshifts and writes data generated by the Muse S headband using the Mind Monitor application.

data

dataframe of the read data

Type

DataFrame

timeshift(*shift='random'*)

Shifts the index column 'TimeStamp'.

Parameters

shift (*None*/'random', *pd.Timestamp* or *pd.Timedelta*) – If shift is not specified, shifts the data by a random time interval between one month and two years to the past.

If shift is a timedelta, shifts the data by that timedelta.

If shift is a timestamp, shifts the data such that the earliest entry is at that timestamp and the remaining values keep the same time distance to the first entry.

write(path)

Writes the dataframe back into a csv file.

Parameters

path (*str*) – path to the write file.

2.3.7 devicely.TimeStampReader

TimeStamp for Android allows you to record the timestamp of an event at the time it occurs. It also allows you to create specific tags such as “Running” or “Walking” and timestamp those specific activities.

class devicely.time_stamp.TimeStampReader(*path*)

Read, timeshift and write data generated by the Android app TimeStamp

data

DataFrame with datetime index and ‘tag’ column.

Type

DataFrame

timeshift (*shift*=‘random’)

Timeshift the data by shifting the index.

Parameters

shift (*None*/‘random’, *pd.Timestamp* or *pd.Timedelta*) – If shift is not specified, shifts the ‘time’ column by a random time interval between one month and two years to the past.

If shift is a timedelta, shift the index by that timedelta.

If shift is a timestamp, shifts the data such that the earliest entry has that timestamp. The remaining values will maintain the same time difference to the first entry.

write(path)

Write the DataFrame stored in ‘data’ to ‘path’ in the same format as it was read.

Parameters

path (*str*) – Path to writing csv. Writing mode: ‘w’

2.4 How to contribute

Whether you would like to add a new sensor, fix a bug or help with packaging, we would love for you to contribute. To make a contribution, please fork [our repository](#) and open a pull request when you are done with your changes.

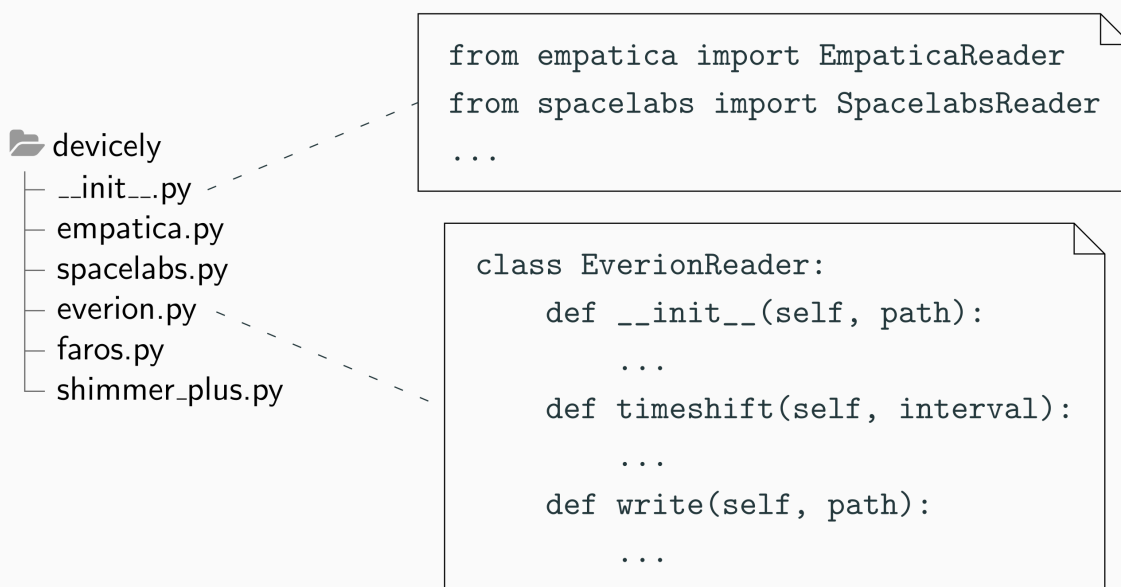
2.4.1 Getting started with development

If you wonder why we do not have a `setup.py`, `setup.cfg` or `requirements.txt`, it is because we use `poetry` for packaging, building and dependency management. To get a development environment, clone the repository and execute `poetry install`. This will create a virtual environment for the project and install all runtime- and development dependencies. Now you can run the tests with `poetry run pytest`, work on the example jupyter notebook with `poetry run jupyter notebook` or enter the virtual environment with `poetry shell`.

2.4.2 Add a sensor class

One reason why you might want to contribute to devicely is to add a new sensor class to the package. Please follow these steps if that is the case:

Your sensor class needs to have its own module in the devicely package:



Please create a class member for each signal that your sensor records. To provide an example, if your sensor contains heart rate and acceleration data, use uppercase attribute names like `reader.ACC` and `reader.HR`. These attributes should be dataframes, indexed by time of measurement. For metadata, you can use basic data structures such as dictionaries. Apart from the individual signal dataframes, your sensor should have an attribute `reader.data` which is a joined dataframe of all individual signal dataframes.

The `timeshift` method should accept three types of parameters: nothing, a `pandas.Timestamp` or a `pandas.Timedelta`. With no parameter, all time-related data attributes are shifted between one month and two years to the past. With a `pandas.Timedelta`, all attributes are shifted by adding the `timedelta` to them. With a `pandas.Timestamp`, all attributes are shifted such that the provided timestamp is the earliest data entry and all other data entries keep the same distance to it.

Timeshifting is not the only way to anonymize data. If your sensor uses other metadata such as a patient id, please add a method `deidentify` to clear such metadata. You can look at `devicely.SpacelabsReader.deidentify` for an example.

With the `reader.write` method, users can write the identified data to be persistent, ideally in the same original sensor format. Keeping the original format is not strictly necessary, just make sure that your reader class can be instantiated with the written data.

2.4.3 Write unit tests

Unit tests ensure that our sensor classes work the way we expect them to, which is why we aim for a high test coverage. The existing test cases are a good example to see how your test case should be structured. In general, have one test case for your sensor and one directory with data for testing. Write test methods for the most important use cases of your sensor, e.g. reading, timeshifting and writing. Please make sure that your test cases run fairly quickly as this helps us run tests locally and keep the GitHub Actions jobs fast. It helps to use small sensor files for testing, e.g. only 30 seconds long.

2.4.4 Write documentation

At devicely, we live by the motto **Docs or it didn't happen**.

Therefore, all sensor classes need to be documented well. Most importantly, you need to add the following docstrings to your class:

1. a docstring on top of the class definition containing information about the sensor and what the most important class attributes are
2. a docstring for each method that users are meant to call specifying the syntax, parameters and return values

Apart from docstrings, all you need to do is add example code to the [notebook with examples](#). Not only is this notebook meant to be run by users themselves, but it is also rendered in our *[Start guide for the devicely package](#)* documentation section.

The docstrings and the notebook are basically all you have to do to document your sensor class. Feel free to look at the existing sensors as a guide!

2.4.5 Provide example data

If you want people to try out your reader class, you need to provide example data. For this purpose we maintain this [example repository with examples and data](#), to which you can create a PR to supply example data for your sensor.

2.4.6 License and Copyright

Currently **devicely** is licensed under the [MIT license](#) and all copyright is attributed to the Digital Health Center (Hasso Plattner Institute).

PYTHON MODULE INDEX

d

`devicely.empatica`, [22](#)
`devicely.everion`, [26](#)
`devicely.faros`, [25](#)
`devicely.muse`, [28](#)
`devicely.shimmer_plus`, [28](#)
`devicely.spacelabs`, [24](#)
`devicely.time_stamp`, [29](#)

A

ACC (*devicely.empatica.EmpaticaReader* attribute), 22
 ACC (*devicely.faros.FarosReader* attribute), 25

B

BVP (*devicely.empatica.EmpaticaReader* attribute), 23

D

data (*devicely.empatica.EmpaticaReader* attribute), 23
 data (*devicely.everion.EverionReader* attribute), 26
 data (*devicely.faros.FarosReader* attribute), 25
 data (*devicely.muse.MuseReader* attribute), 28
 data (*devicely.shimmer_plus.ShimmerPlusReader* attribute), 28
 data (*devicely.spacelabs.SpacelabsReader* attribute), 24
 data (*devicely.time_stamp.TimeStampReader* attribute), 29
 default_feature_tags (*devicely.everion.EverionReader* attribute), 27
 default_sensor_tags (*devicely.everion.EverionReader* attribute), 27
 default_signal_tags (*devicely.everion.EverionReader* attribute), 27
 deidentify() (*devicely.spacelabs.SpacelabsReader* method), 24
 devicely.empatica module, 22
 devicely.everion module, 26
 devicely.faros module, 25
 devicely.muse module, 28
 devicely.shimmer_plus module, 28
 devicely.spacelabs module, 24
 devicely.time_stamp module, 29

E

ECG (*devicely.faros.FarosReader* attribute), 25
 EDA (*devicely.empatica.EmpaticaReader* attribute), 23
 EmpaticaReader (class in *devicely.empatica*), 22
 EverionReader (class in *devicely.everion*), 26

F

FarosReader (class in *devicely.faros*), 25
 FEATURE_TAGS (*devicely.everion.EverionReader* attribute), 27

H

HR (*devicely.empatica.EmpaticaReader* attribute), 23
 HRV (*devicely.faros.FarosReader* attribute), 25

I

IBI (*devicely.empatica.EmpaticaReader* attribute), 22

J

join_dataframes() (*devicely.faros.FarosReader* method), 25

M

Marker (*devicely.faros.FarosReader* attribute), 25
 metadata (*devicely.spacelabs.SpacelabsReader* attribute), 24
 module
 devicely.empatica, 22
 devicely.everion, 26
 devicely.faros, 25
 devicely.muse, 28
 devicely.shimmer_plus, 28
 devicely.spacelabs, 24
 devicely.time_stamp, 29
 MuseReader (class in *devicely.muse*), 28

S

sample_freqs (*devicely.empatica.EmpaticaReader* attribute), 22
 sample_freqs (*devicely.faros.FarosReader* attribute), 25

SENSOR_TAGS (*devicely.everion.EverionReader* attribute), 27
ShimmerPlusReader (*class in devicely.shimmer_plus*), 28
SIGNAL_TAGS (*devicely.everion.EverionReader* attribute), 26
SpacelabsReader (*class in devicely.spacelabs*), 24
start_time (*devicely.faros.FarosReader* attribute), 25
start_times (*devicely.empatica.EmpaticaReader* attribute), 22
subject (*devicely.spacelabs.SpacelabsReader* attribute), 24

T

TEMP (*devicely.empatica.EmpaticaReader* attribute), 23
timeshift() (*devicely.empatica.EmpaticaReader* method), 23
timeshift() (*devicely.everion.EverionReader* method), 27
timeshift() (*devicely.faros.FarosReader* method), 25
timeshift() (*devicely.muse.MuseReader* method), 28
timeshift() (*devicely.shimmer_plus.ShimmerPlusReader* method), 28
timeshift() (*devicely.spacelabs.SpacelabsReader* method), 24
timeshift() (*devicely.time_stamp.TimeStampReader* method), 29
TimeStampReader (*class in devicely.time_stamp*), 29

U

units (*devicely.faros.FarosReader* attribute), 25
units (*devicely.shimmer_plus.ShimmerPlusReader* attribute), 28

V

valid_measurements (*devicely.spacelabs.SpacelabsReader* attribute), 24

W

write() (*devicely.empatica.EmpaticaReader* method), 23
write() (*devicely.everion.EverionReader* method), 27
write() (*devicely.faros.FarosReader* method), 26
write() (*devicely.muse.MuseReader* method), 29
write() (*devicely.shimmer_plus.ShimmerPlusReader* method), 28
write() (*devicely.spacelabs.SpacelabsReader* method), 24
write() (*devicely.time_stamp.TimeStampReader* method), 29