

---

# **pymepix Documentation**

***Release 1.1.dev0***

**CFEL Controlled Molecule Imaging group**

**May 31, 2023**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Getting started</b>	<b>5</b>
2.1	Installing . . . . .	5
2.1.1	Installing from PyPI (platform-independent) . . . . .	5
2.1.2	Installing from git source directly (platform-independent) . . . . .	5
2.1.3	Build Documentation . . . . .	5
2.2	Dependencies . . . . .	5
<b>3</b>	<b>Connecting and Configuring</b>	<b>7</b>
3.1	Connecting . . . . .	7
3.2	Configuring . . . . .	7
<b>4</b>	<b>Acquisition</b>	<b>9</b>
4.1	Polling . . . . .	9
4.2	Callback . . . . .	10
4.3	Pipelines . . . . .	10
<b>5</b>	<b>Data Formats</b>	<b>13</b>
5.1	UDP Packets . . . . .	13
5.2	Decoded Pixels . . . . .	13
5.3	Decoded Triggers . . . . .	14
5.4	Time of Flight/Event . . . . .	14
5.5	Centroid Data . . . . .	14
<b>6</b>	<b>Examples</b>	<b>17</b>
<b>7</b>	<b>PymepixAcq - Command line</b>	<b>21</b>
<b>8</b>	<b>Pymepix postprocessing</b>	<b>23</b>
<b>9</b>	<b>Troubleshooting</b>	<b>25</b>
<b>10</b>	<b>Pymepix Developer documentation</b>	<b>27</b>
10.1	API reference . . . . .	27
10.1.1	General overview . . . . .	27
10.1.1.1	pymepix . . . . .	27
10.1.1.2	timepixdevice and timepixdef . . . . .	27

10.1.1.3	config module . . . . .	27
10.1.1.4	core module . . . . .	28
10.1.1.5	processing module . . . . .	28
10.1.1.6	SPIDR module . . . . .	28
10.1.1.7	util module . . . . .	28
10.1.1.8	Class overview . . . . .	28
10.1.2	pymepix . . . . .	28
10.1.2.1	pymepix package . . . . .	28
<b>11</b>	<b>References</b>	<b>67</b>
<b>12</b>	<b>Pymepix Documentation</b>	<b>69</b>
	<b>Bibliography</b>	<b>71</b>
	<b>Python Module Index</b>	<b>73</b>
	<b>Index</b>	<b>75</b>

Pymepix documentation



# CHAPTER 1

---

## Introduction

---

Pymepix is intended to bridge the gap between Timepix3 and Python. The goal of the library is to allow a user without deep technical knowledge of Timepix3 to establish a connection, start acquisition, and retrieve and plot pixel and timing information in as few lines of code as possible; at the same time it provides all details to unleash the full power of Timepix3-SPIDR hardware. This is achieved by classes that act as a black-box, handling all of the low level TCP communication and decoding of the UDP data-stream, presenting them in a pythonic fashion. More advanced and lower-level control of SPIDR and Timepix3 is still available from these black-box classes or can be established directly by the user. For easy installation, it only depends on the standard python library, numpy and scikit-learn.





## 2.1 Installing

### 2.1.1 Installing from PyPI (platform-independent)

Execute `pip install pymepix`. This should install pymepix including all dependencies.

### 2.1.2 Installing from git source directly (platform-independent)

You can clone pymepix from our main git repository:

```
git clone https://github.com/CFEL-CMI/pymepix.git
```

Navigate into the pymepix library (`cd pymepix`) and run `pip install .`

### 2.1.3 Build Documentation

To build the documentation for pymepix locally perform the following commands. The first line is only required if there are changes in the package structure or new classes or packages have been added. To only build the existing documentation only the second line must be executed.

```
1 sphinx-apidoc -o ./doc/source/ ./pymepix
2 python setup.py build_sphinx
```

Adapt `pymepix/config/default.yaml` according to your setup.

## 2.2 Dependencies

The majority of pymepix only depends on numpy. To use centroiding, the scikit-learn package is required

- *numpy*
- *scikit-learn*: Centroiding and data reduction (Using DBSCAN algorithm for clustering)
- *scipy*: Calculation of the centroids properties from the identified clusters
- *pyzmq*: Inter process communication in the processing pipeline
- *h5py*: Saving processed data as hdf5 files
- *tqdm*: Display a progressbar for post processing
- *pyyaml*: Konfiguration of camera (ip, port, ...)
- *pyserial* (optional): Only used for inclusion of USBTrainID at FLASH and XFEL

---

## Connecting and Configuring

---

### 3.1 Connecting

For the camera to work you will have to set up the IP address on your machine, that the camera then communicates with. For Timepix3 with 10 Gb/s that is 192.168.100.1. Look up the official documentation for your camera to find out more.

**Before using Pymepix, make sure your camera works properly with the SoPhy software.**

The IP address of your TPX camera is the one seen on the OLED screen. Connecting to SPIDR can be done with:

```
>>> timepix = Pymepix(('192.168.100.10', 50000))
```

The number of devices can be found using:

```
>>> len(timepix)
1
```

Meaning we have one device. To access this device directly, use:

```
tpx0 = timepix[0]
```

And to check the device name:

```
>>> tpx0.deviceName
W0026_K08
```

### 3.2 Configuring

To set the biasVoltage to 50 Volts in spidr you can do:

```
>>> timepix.biasVoltage = 50
```

Setting the we can manage its settings directly. To easily setup the device we can use a SoPhy config file (.spx):

```
tpx0.loadConfig('myFile.spx')
```

This sets up all the DAC setting and pixel configurations. Individual parameters can also be set for example. To set the fine threshold to 100 mV do:

```
>>> tpx0.Vthreshold_fine = 100
```

pixel threshold configurations can be set by passing a 256x256 numpy array:

```
import numpy as np
tpx0.pixelThreshold[...] = 0
```

The same for pixel masks, to set a checkboard mask do:

```
tpx0.pixelMask[:,2] = 1
```

These need to be uploaded to timepix before they take effect:

```
>>> tpx0.uploadPixels()
```

The full list of parameters that can be set can be found in `timepixdevice()`.

# CHAPTER 4

## Acquisition

Acquisition can be started and stopped by:

```
1 import time
2 from pymepix import Pymepix
3
4 #Connect
5 timepix = Pymepix(('192.168.1.10',50000))
6
7 #Start acquisition
8 timepix.start()
9
10 #Wait
11 time.sleep(1.0)
12
13 #Stop acquisition
14 timepix.stop()
```

Pymepix provides data as a tuple given by (MessageType,data). These are explained in *Data Formats*. Retrieving the data can be done in to ways: Polling or Callback

### 4.1 Polling

Polling is where pymepix will place anything retrieved from Timepix into a ring polling buffer. This is the default mode but to reenale it you can use:

```
>>> timepix.enablePolling(maxlen=1000)
```

where *maxlen* describes the maximum number of elements in the buffer before older values are overwritten.

The user can retrieve this data by using:

```
>>> timepix.poll()
(MessageType.RawData, (array[98732405897234589802345, dtype=uint8],12348798))
```

If there is nothing in the polling buffer then a `PollBufferEmpty` exception is raised. The poll buffer is limited in size but can be extended by doing:

```
>>> timepix.pollBufferLength = 5000
```

This will clear all objects using the polling buffer.

## 4.2 Callback

The callback method allows the user to deal with the data immediately when it is received. Setting this will clear the polling buffer of any contents.

To set a callback, first you need a function, for example:

```
def my_callback(data_type, data):  
    print('My callback is running!!!!')
```

The format of the function must accept two parameters, `MessageType` and an extra data parameter. These are explained in [Data Formats](#). Now to make pymepix use it simply do:

```
>>> timepix.dataCallback = my_callback
```

Now when acquisition is started:

```
>>> timepix.start()
```

The output seen is:

```
.. code-block:: sh
```

```
My callback is running!!!! My callback is running!!!! My callback is running!!!! My callback is run-  
ning!!!! My callback is running!!!!
```

## 4.3 Pipelines

Pymepix uses pipeline objects in order to process data. Each pipeline is set for each timepix device so each timepix can have a different data pipeline. You can configure them to postprocess or output data in certain ways. For example the `PixelPipeline` object will read from a UDP packet stream and decode the stream into *pixel x*, *pixel y*, *time of arrival* and *time over threshold* arrays. All data is propagated forward through the pipeline so both UDP packets and decoded pixels are output.

To use the (default) `PixelPipeline` pipeline on the first connected timepix device you can do:

```
from pymepix.processing import PixelPipeline, CentroidPipeline  
  
timepix[0].setupAcquisition(PixelPipeline)
```

If you need centroid you instead can do:

```
>>> timepix[0].setupAcquisition(CentroidPipeline)
```

Configuring the pipelines can be done using the acquisition property for the timepix device, for example to enable TOFs you can do:

```
>>> timepix[0].acquisition.enableEvents = True
```

A list of pipelines and setting can be found in `acquisition()`





Contains a list of possible data formats output during acquisition. Each entry of the data section represents another element in the tuple. Example shows how to read the data through polling

### 5.1 UDP Packets

**Data Type:** `MessageType.RawData`

**Data:**

**`array(uint64)`** list of UDP packets

**`uint64`** global timer from Timepix at time packets were recieved

Example:

```
1 data_type, data = timepix.poll()
2 if data_type is MessageType.RawData:
3     packets, longtime = data
```

### 5.2 Decoded Pixels

**Data Type:** `MessageType.PixelData`

**Data:**

**`array(uint64)`** pixel x position

**`array(uint64)`** pixel y position

**`array(float)`** global time of arrival in seconds

**`array(uint64)`** time over threshold in nanoseconds

Example:

```
1 data_type, data = timepix.poll()
2 if data_type is MessageType.PixelData:
3     x, y, toa, tot = data
```

## 5.3 Decoded Triggers

**Data Type:** `MessageType.TriggerData`

**Data:**

- array(uint64)** trigger number
- array(float)** global trigger time in seconds

Example:

```
1 data_type, data = timepix.poll()
2 if data_type is MessageType.TriggerData:
3     t_num, t_time = data
```

## 5.4 Time of Flight/Event

**Data Type:** `MessageType.EventData`

**Data:**

- array(uint64)** trigger number
- array(uint64)** pixel x position
- array(uint64)** pixel y position
- array(float)** time of flight relative to its trigger in seconds
- array(uint64))** time over threshold in nanoseconds

Example:

```
1 data_type, data = timepix.poll()
2 if data_type is MessageType.EventData:
3     trigger, x, y, tof, tot = data
```

## 5.5 Centroid Data

**Data Type:** `MessageType.CentroidData`

**Data:**

- array(uint64)** trigger number
- array(float)** center of mass x position
- array(float)** center of mass y position
- array(float)** minimum cluster time of flight
- array(float)** average cluster time over threshold
- array(uint64)** maximum cluster time over threshold

**array(uint64)** cluster size

Example:

```
1 data_type, data = timepix.poll()
2 if data_type is MessageType.CentroidData:
3     trigger, x, y, tof, avg_tot, max_tot, size = data
```



## CHAPTER 6

---

### Examples

---

Starting timepix and polling data:

```
import pymepix
from pymepix.processing import MessageType
import numpy as np

#Connect to SPIDR
timepix = pymepix.pymepix_connection.PymepixConnection(('192.168.1.10', 50000))

#Set bias voltage
timepix.biasVoltage = 50

#Set pixel masks
timepix[0].pixelThreshold = np.zeros(shape=(256,256), dtype=np.uint8)
timepix[0].pixelMask = np.zeros(shape=(256,256), dtype=np.uint8)
timepix[0].uploadPixels()

#Start acquisition
timepix.start()

while True:
    try:
        #Poll
        data_type,data = timepix.poll()
    except pymepix.PollBufferEmpty:
        #If empty then just loop
        continue

    #Handle Raw
    if data_type is MessageType.RawData:

        print('UDP PACKET')

        packets,longtime = data
```

(continues on next page)

(continued from previous page)

```

    print('Packet ',packets)
    print('Time', longtime)

    #Handle Pixels
    elif data_type is MessageType.PixelData:

        print('I GOT PIXELS!!!!')

        x,y,toa,tot = data

        print('x',x)
        print('y', y)
        print('toa', toa)
        print('tot',tot)

#Stop
timepix.stop()

```

Using callbacks to acquire:

```

import pymepix
from pymepix.processing import MessageType
import numpy as np
import time

#Connect to SPIDR
timepix = pymepix.Pymepix(('192.168.1.10',50000))

#Set bias voltage
timepix.biasVoltage = 50

#Set pixel masks
timepix[0].pixelThreshold = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].pixelMask = np.zeros(shape=(256,256),dtype=np.uint8)
timepix[0].uploadPixels()

#Define callback
def my_callback(data_type,data):
    print('MY CALLBACK!!!!')
    #Handle Raw
    if data_type is MessageType.RawData:

        print('UDP PACKET')

        packets,longtime = data

        print('Packet ',packets)
        print('Time', longtime)

    #Handle Pixels
    elif data_type is MessageType.PixelData:

        print('I GOT PIXELS!!!!')

        x,y,toa,tot = data

```

(continues on next page)

(continued from previous page)

```
    print('x',x)
    print('y', y)
    print('toa', toa)
    print('tot',tot)

#Set callback
timepix.dataCallback = my_callback

#Start acquisition
timepix.start()
#Wait 5 seconds
time.sleep(5.0)
#Stop
timepix.stop()
```





---

## PymepixAcq - Command line

---

Included with pymepix is a command line code using the pymepix library to acquire from timepix. The command line interface

- “connect”: to connect to a running timepix camera and record data
- “post-process”: to post-process recorded raw data files into easier usable hdf5 files containing raw and centroided data

Doing:

```
pymepix-acq --help
```

Outputs the help:

```
usage: pymepix-acq [-h] {connect,post-process} ...

Timepix acquisition script

positional arguments:
  {connect,post-process}
    connect             Connect to TimePix camera and acquire data.
    post-process        Perform post-processing for an acquired raw data file.

optional arguments:
  -h, --help            show this help message and exit
```

You can access the documentation for both commands by executing “pymepix-acq connect -h” or “pymepix-acq post-process -h” respectively.



---

## Pymepix postprocessing

---

The raw data acquired from the camera could be processed from command line with the command. The processing can also be triggered from the *PymepixViewer*.

Doing:

```
pymepix post-process -f FILE -o OUTPUT_FILE [-t TIMEWALK_FILE] [-c CENT_TIMEWALK_
→FILE] [-n NUMBER_OF_PROCESSES]
```

The generated output file has HDF data format may contain the following datagroups in its root:

- **centroided**
- **raw**
- **timing/timepix**
- **triggers**

The **centroided** datagroups contains the data after centroiding processing. It consists of several datasets : “trigger nr”, “x”, “y”, “tof”, “tot avg”, “tot max”, “clustersize”. Where “trigger nr” is event number, “x”/“y” - coordinates of centroid, tof is time-of-flight (time-of-arrival corrected to the timewalk effect), “tot avg” average value of tot for all voxels in the cluster, “tot max” - max tot value, “clustersize” - the number of voxels in the detected cluster.

The **raw** datagroups contains event data - voxel data with tof synchronized to first trigger. it consists of following datasets: “trigger nr”, “x”, “y”, “tof”, “tot”.

The **timing/timepix** datagroup has only two datasets: “trigger nr”, “timestamp”. Where “trigger nr” contains triggering event numbers from first trigger, while dataset “time” contains the timestamps for the corresponding trigger event in nanosecond in absolute time from the timer of the camera.

Datagroup **triggers** may contain two subgroups “trigger1” and “trigger2” corresponding to the first and second trigger of the camera. Each subgroup consists of only one dataset “time”. These are firing times of the corresponding trigger starting from acquisition in seconds. In case of first trigger these are the times of rising front of the detected trigger pulse. For the second trigger both rising and falling pulse edges are detected. Negative values correspond to the falling edge.

Here’s an example to retrieve the data from the HDF5 file into a Pandas DataFrame:



---

### Troubleshooting

---

- **Whenever there are problems when working with the camera** First make sure you can ping the Timepix camera to ensure a working connection.  
Next try starting the SoPhy software and see if it can communicate properly with the camera. Remember to close SoPhy afterwards, as there can only be one process using the address.
- **Make sure to load the correct config file.** If the parameters are off, you might not be able to see anything.
- **Use a flashlight!** When the camera is properly connected and set up, you may use a flashlight to shine directly into the lens and next to it in quick succession.
- **You can see ToA but no ToF data** Check and maybe reconfigure the trigger.
- **Error: Address is already in use** If you get this error, look for any other process that is running and uses the corresponding IP and port. Also try restarting the camera
- [genindex](#)
- [modindex](#)
- [search](#)



This developer documentation contains the API reference for *pymepix*.

## 10.1 API reference

### 10.1.1 General overview

The main Pymepix library is built up in several different submodules which each tackle a different task in working with the Timepix camera. As seen in the API index those are *\* config \* core \* processing \* SPIDR \* util*

The top layer Pymepix consists of *pymepix*, *timepixdef* and *timepixdevice*.

#### 10.1.1.1 pymepix

*pymepix* provides the highest level of interaction with the library. A single *Pymepix* object will hold all connected Timepix devices and manage the users' interaction with those.

#### 10.1.1.2 timepixdevice and timepixdef

A *timepixdevice* object holds all the communication with a single camera. It basically configures, starts and stops. The *timepixdef* has multiple enums to encode all kinds of parameters for Timepix.

#### 10.1.1.3 config module

The *config* module gets the information for config parameters.

*timepixconf* is the base class for the possible configurations.

*defaultconfig* holds hardcoded config parameters for the camera to initialize.

*sophyconfig* imports information from SoPhy (.spx) config files. It reads and transforms that information to be used by Pymepix.

#### 10.1.1.4 core module

The *core* module consists of only the log class. It defines functionality for Pymepix' needs and uses the basic python logging module.

#### 10.1.1.5 processing module

The *processing* module provides the data pipeline to process the incoming camera data. Pymepix can use different acquisition pipelines to process the data. Those are defined in *acquisition* with the base functionality provided by *baseacquisition*. An acquisition pipeline determines which steps work in what order on the incoming data and connects those.

Each pipeline consists of acquisition stages (*baseacquisition*), where one stage holds the information about one logical step in the pipeline. Those tasks are currently *udpsampler* (capturing the packets), *rawtodisk* (saving the raw data), *pipeline\_packet\_processor* (interpreting the raw packets) and *pipeline\_centroid\_calculator* (compress data by finding blob centers). Each of these specific pipeline steps overwrites the *BasePipelineObject*, which is in fact a python *multiprocessing.Process*.

The majority of the logic for the *pipeline\_packet\_processor* and the *pipeline\_centroid\_calculator* is separated in the classes *centroid\_calculator* and *packet\_processor*. The *pipeline\_* classes only add functionality for the integration of those classes into the multiprocessing pipeline.

Each stage knows the task it has to fulfill and then creates one or multiple processes to work on that task in parallel.

*datatypes* provides an enum to classify the data that is passed through the pipeline at each step.

#### 10.1.1.6 SPIDR module

This module communicates with the SPIDR chip of the Timepix. One *spidrcontroller* knows about one or more *spidrdevices*. *spidrcmds* lists the known commands to pass information and instructions to a chip. *spidrdefs* extends those commands by constants that can be passed. *error* contains information on possible errors from SPIDR.

#### 10.1.1.7 util module

*storage* provides some functionality to save data.

*spidrDummyTCP* and *-UDP* can be used to simulate a timepix camera. Both are still rudimentary but helpful for debugging.

*spidrDummyTCP* accepts packets in so the configuration of timepix can be tested.

*spidrDummyUDP* samples and sends packets from a given file into the void. This can be used to test the pipeline functionality by capturing those packets with Pymepix.

#### 10.1.1.8 Class overview

### 10.1.2 pymepix

#### 10.1.2.1 pymepix package

##### Subpackages

##### pymepix.SPIDR package



## Submodules

### pymepix.SPIDR.error module

**exception** pymepix.SPIDR.error.PymePixException(*error\_code*)

Bases: Exception

ERR\_STR = ['no error', 'ERR\_UNKNOWN\_CMD', 'ERR\_MSG\_LENGTH', 'ERR\_SEQUENCE', 'ERR\_ILLEG

MONITOR\_ERR\_STR = ['MON\_ERR\_TEMP\_DAQ', 'MON\_ERR\_POWER\_DAQ']

SPIDR\_ERR\_STR = ['SPIDR\_ERR\_I2C\_INIT', 'SPIDR\_ERR\_LINK\_INIT', 'SPIDR\_ERR\_MPL\_INIT', 'S

STORE\_ERR\_STR = ['no error', 'STORE\_ERR\_TPX', 'STORE\_ERR\_WRITE', 'STORE\_ERR\_WRITE\_CHECK

TPX3\_ERR\_STR = ['no error', 'TPX3\_ERR\_SC\_ILLEGAL', 'TPX3\_ERR\_SC\_STATE', 'TPX3\_ERR\_SC\_E

errorMessage(*code*)

**class** pymepix.SPIDR.error.SPIDRErrorDefs

Bases: object

ERR\_ADC\_HARDW = 7

ERR\_DAC\_HARDW = 8

ERR\_FLASH\_STORAGE = 10

ERR\_ILLEGAL\_PAR = 4

ERR\_MONITOR = 11

ERR\_MON\_HARDW = 9

ERR\_MSG\_LENGTH = 2

ERR\_NONE = 0

ERR\_NOT\_IMPLEMENTED = 5

ERR\_SEQUENCE = 3

ERR\_TPX3\_HARDW = 6

ERR\_UNKNOWN\_CMD = 1

### pymepix.SPIDR.spidrcmds module

This module contains a list of all (found) commands for the SPIDR board

**class** pymepix.SPIDR.spidrcmds.SpidrCmds

Bases: enum.IntEnum

A class that packages all the commands under a single name

CMD\_AUTOTRIG\_START = 1090

CMD\_AUTOTRIG\_STOP = 1091

CMD\_BIAS\_SUPPLY\_ENA = 1375

CMD\_BURN\_EFUSE = 297

CMD\_CLEAR\_BUSY = 2313

CMD\_CONFIG\_CTPR = 288

```
CMD_DDRIVEN_READOUT = 1094
CMD_DECODERS_ENA = 1377
CMD_DISPLAY_INFO = 2315
CMD_ERASE_ADDRPORTS = 1652
CMD_ERASE_DACS = 1653
CMD_ERASE_PIXCONF = 1655
CMD_ERASE_REGISTERS = 1654
CMD_GET_ADC = 1352
CMD_GET_AVDD = 1355
CMD_GET_AVDD_NOW = 1357
CMD_GET_BOARDID = 2318
CMD_GET_CHIPBOARDID = 2319
CMD_GET_CTPR = 290
CMD_GET_DAC = 282
CMD_GET_DEVICECOUNT = 2317
CMD_GET_DEVICEID = 272
CMD_GET_DEVICEIDS = 273
CMD_GET_DEVICEPORT = 278
CMD_GET_DVDD = 1356
CMD_GET_DVDD_NOW = 1359
CMD_GET_EFUSES = 296
CMD_GET_EXTSHUTTERCNTR = 1366
CMD_GET_FANSPEED = 1385
CMD_GET_FIRMWVERSION = 2306
CMD_GET_FPGATEMP = 1384
CMD_GET_GENCONFIG = 820
CMD_GET_GPIO = 1920
CMD_GET_HEADERFILTER = 2309
CMD_GET_HUMIDITY = 1390
CMD_GET_IPADDR_DEST = 276
CMD_GET_IPADDR_SRC = 274
CMD_GET_LOCALTEMP = 1354
CMD_GET_OUTBLOCKCONFIG = 828
CMD_GET_PIXCONF = 557
CMD_GET_PLLCONFIG = 822
CMD_GET_PRESSURE = 1391
```

```
CMD_GET_PWRPULSECONFIG = 1371
CMD_GET_READOUTSPEED = 1380
CMD_GET_REMOTETEMP = 1353
CMD_GET_SERVERPORT = 279
CMD_GET_SHUTTERCNTR = 1367
CMD_GET_SHUTTEREND = 1365
CMD_GET_SHUTTERSTART = 1364
CMD_GET_SLVSCONFIG = 830
CMD_GET_SOFTWVERSION = 2305
CMD_GET_SPIDRREG = 1923
CMD_GET_SPIDR_ADC = 1358
CMD_GET_STARTOPTS = 1661
CMD_GET_TIMER = 1362
CMD_GET_TPNUMBER = 819
CMD_GET_TPPERIODPHASE = 816
CMD_GET_TRIGCONFIG = 1088
CMD_GET_VDD = 1388
CMD_GET_VDD_NOW = 1389
CMD_MASK = 65535
CMD_NOP = 0
CMD_NOREPLY = 524288
CMD_PAUSE_READOUT = 1095
CMD_PWRPULSE_ENA = 1373
CMD_READ_FLASH = 1662
CMD_REINIT_DEVICE = 294
CMD_REINIT_DEVICES = 295
CMD_REPLY = 65536
CMD_RESET_COUNTERS = 1368
CMD_RESET_DEVICE = 292
CMD_RESET_DEVICES = 293
CMD_RESET_MODULE = 2311
CMD_RESET_PIXELS = 558
CMD_RESET_TIMER = 1361
CMD_RESTART_TIMERS = 1360
CMD_SELECT_CHIPBOARD = 1387
CMD_SEQ_READOUT = 1093
```

```
CMD_SET_BIAS_ADJUST = 1376
CMD_SET_BOARDID = 1926
CMD_SET_BUSY = 2312
CMD_SET_CHIPBOARDID = 1925
CMD_SET_CTPR = 289
CMD_SET_CTPR_LEON = 291
CMD_SET_DAC = 283
CMD_SET_DACS_DFLT = 287
CMD_SET_EXTDAC = 826
CMD_SET_FANSPEED = 1386
CMD_SET_GENCONFIG = 821
CMD_SET_GPIO = 1921
CMD_SET_GPIO_PIN = 1922
CMD_SET_HEADERFILTER = 2310
CMD_SET_IPADDR_DEST = 277
CMD_SET_IPADDR_SRC = 275
CMD_SET_LOGLEVEL = 2314
CMD_SET_OUTBLOCKCONFIG = 829
CMD_SET_OUTPUTMASK = 1378
CMD_SET_PIXCONF = 554
CMD_SET_PLLCONFIG = 823
CMD_SET_PWRPULSECONFIG = 1372
CMD_SET_READOUTSPEED = 1379
CMD_SET_SENSEDAC = 824
CMD_SET_SERVERPORT = 281
CMD_SET_SLVSCONFIG = 831
CMD_SET_SPIDRREG = 1924
CMD_SET_TIMEOFDAY = 2316
CMD_SET_TIMER = 1363
CMD_SET_TPNUMBER = 818
CMD_SET_TPPERIODPHASE = 817
CMD_SET_TRIGCONFIG = 1089
CMD_STORE_ADDRPORTS = 1648
CMD_STORE_DACS = 1649
CMD_STORE_PIXCONF = 1651
CMD_STORE_REGISTERS = 1650
```

```

CMD_STORE_STARTOPTS = 1660
CMD_T0_SYNC = 1381
CMD_TPX_POWER_ENA = 1374
CMD_UPLOAD_PACKET = 827
CMD_VALID_ADDRPORTS = 1656
CMD_VALID_DACS = 1657
CMD_VALID_PIXCONF = 1659
CMD_VALID_REGISTERS = 1658
CMD_WRITE_FLASH = 1663

```

## pymepix.SPIDR.spidrcontroller module

SPIDR related classes

**class** pymepix.SPIDR.spidrcontroller.**SPIDRController**(*dst\_ip\_port, src\_ip\_port*)

Bases: [pymepix.core.log.Logger](#)

Object that interfaces over ethernet with the SPIDR board

This object interfaces with the spidr board through TCP and is used to send commands and receive data. It can be treated as a list of SpidrDevice objects to talk to a specific device

### Parameters

- **dst\_ip\_port** (tuple of str and int) – socket style tuple of SPIDR ip address and port
- **src\_ip\_port** (tuple of str and int, optional) – socket style tuple of the IP address and port of the interface that is connecting to SPIDR

## Examples

The class can be used to talk to SPIDR

```

>>> spidr = SPIDRController(('192.168.1.10', 50000))
>>> spidr.fpgaTemperature
39.5

```

Or access a specific SpidrDevice (e.g. Timepix/Medipix)

```

>>> spidr[0].deviceId
7272
>>> spidr[1].deviceId
2147483648

```

**Warning:** This object assumes SPIDR is working as intended however since this is still in development there are a few functions that do not behave as they should, this will be documented in their relevant areas.

### CpuToTpx

Cpu2Tpx register access

**Parameters** `value` (*int*) – Value to write to the register

**Returns** Current value of the register

**Return type** `int`

**Raises** `PymePixException` – Communication error

## Notes

Register controls clock setup

### DeviceAndPorts

#### ShutterTriggerCount

Number of times the shutter is triggered in auto trigger mode

**Parameters** `value` (*int*) – Trigger count to set for auto trigger mode ( Set to 0 for infinite triggers)

**Returns** Current value of the trigger count read from SPIDR

**Return type** `int`

**Raises** `PymePixException` – Communication error

#### ShutterTriggerCtrl

Shutter Trigger Control register access

**Parameters** `value` (*int*) – Value to write to the register

**Returns** Current value of the register

**Return type** `int`

**Raises** `PymePixException` – Communication error

#### ShutterTriggerDelay

Delay time before shutter can be triggered again in auto trigger mode

**Parameters** `value` (*int*) – Time in ns

**Returns** `value` – Current time in ns read from SPIDR

**Return type** `int`

**Raises** `PymePixException` – Communication error

#### ShutterTriggerFreq

Triggering frequency for the auto trigger

**Parameters** `value` (*float*) – Frequency in mHz

**Returns** Frequency value in mHz read from SPIDR

**Return type** `float`

**Raises** `PymePixException` – Communication error

#### ShutterTriggerLength

Length of time shutter remains open at each trigger

**Parameters** `value` (*int*) – Length in ns

**Returns** `value` – Current length in ns read from SPIDR

**Return type** `int`

**Raises** `PymePixException` – Communication error

### **ShutterTriggerMode**

Controls how the shutter is triggered

**Parameters** `value` (`SpidrShutterMode`) – Shutter trigger mode to set

**Returns** Current shutter operation mode read from SPIDR

**Return type** `SpidrShutterMode`

**Raises** `PymePixException` – Communication error

### **Notes**

AutoTrigger is the only functioning trigger mode that SPIDR can operate in

### **TdcTriggerCounter**

Trigger packets sent by SPIDR since last counter reset

### **UdpMonPacketCounter**

### **UdpPacketCounter**

UDP packets sent by SPIDR since last counter reset

### **UdpPausePacketCounter**

UDP packets collected during readout pause since last counter reset

### **avdd**

### **avddNow**

### **biasVoltage**

Bias voltage

**Parameters** `volts` (`int`) – Bias voltage to supply in volts Minimum is 12V and Maximum is 104V

**Returns** Current bias supply in volts

**Return type** `int`

**Raises** `PymePixException` – Communication error

### **chipboardFanSpeed**

### **chipboardId**

### **clearBusy()**

### **closeShutter()**

Immediately closes the shutter

**Raises** `PymePixException` – Communication error

### **convertHtonl(x)**

### **convertNtohl(x)**

### **datadrivenReadout()**

Set SPIDR into data driven readout mode

Data driven mode refers to the pixels packets sent as they are hit rather than camera style frames

**Raises** `PymePixException` – Communication error

**Warning:** This is the only tested mode for pymepix. It is recommended that this is enabled

**deviceCount**

Count of devices connected to SPIDR

**Returns** Number of devices connected to SPIDR

**Return type** int

**Raises** `PymePixException` – Communication error

**Warning:** SPIDR always returns 4 since it currently can't determine if the devices are actually valid or not

**deviceIds**

The ids of all devices connected to the SPIDR board

**Returns** A list all connected device ids

**Return type** list of int

**Raises** `PymePixException` – Communication error

**Notes**

Index of devices are the same as the those in the `SPIDRController` list

```
>>> spidr[1].deviceId == spidr.deviceIds[1]
True
```

**disableExternalRefClock()**

SPIDR recieves its reference clock internally

This should be set in single SPIDR mode. When combining other SPIDR board, the master will set this to disabled

**Raises** `PymePixException` – Communication error

**disablePeriphClk80Mhz()****dvdd****dvddNow****enableDecoders(enable)**

Determines whether the internal FPGA decodes ToA values

Time of Arrival from UDP packets are gray encoded if this is enabled then SPIDR will decode them for you, otherwise you have to do this yourself after extracting them

**Parameters** **enable** (*bool*) – True - enable FPGA decoding False - disable FPGA decoding

**Raises** `PymePixException` – Communication error

---

**Tip:** Enable this

---



**enableExternalRefClock()**

SPIDR receives its reference clock externally

This is often used when combining multiple Timepixs together so they can synchronize their clocks. The SPIDR board essentially acts as a slave to other SPIDRs

**Raises** `PymePixException` – Communication error

**enablePeriphClk80Mhz()****externalShutterCounter****firmwareVersion**

Firmware version

**Returns** Version number of firmware within the FPGA

**Return type** `int`

**Raises** `PymePixException` – Communication error

**fpgaTemperature**

Temperature of FPGA board read from sensor

**Returns** Temperature in Celsius

**Return type** `float`

**Raises** `PymePixException` – Communication error

**getAdc(channel, nr\_of\_samples)****getSpidrReg(addr)****humidity**

Humidity read from sensor

**Returns** Humidity as percentage

**Return type** `int`

**Raises** `PymePixException` – Communication error

**linkCounts****localTemperature**

Local Temperature read from sensor

**Returns** Temperature in Celsius

**Return type** `float`

**Raises** `PymePixException` – Communication error

**openShutter()**

Immediately opens the shutter indefinitely

**Raises** `PymePixException` – Communication error

**Notes**

This overwrites shutter configurations with one that forces an open shutter

**pauseReadout()****pressure**

Pressure read from sensor

**Returns** Pressure in bar

**Return type** int

**Raises** `PymePixException` – Communication error

**reinitDevices** ()

Resets and initializes all devices

**Raises** `PymePixException` – Communication error

**remoteTemperature**

Remote temperature read from sensor

**Returns** Temperature in Celsius

**Return type** float

**Raises** `PymePixException` – Communication error

**request** (*cmd, dev\_nr, message\_length, expected\_bytes=0*)

Sends a command and (may) receive a reply

**Parameters**

- **cmd** (`SpidrCmds`) – Command to send
- **dev\_nr** (*int*) – Device to send the request to. 0 is SPIDR and device number n is n+1
- **message\_length** (*int*) – Length of the message in bytes
- **expected\_bytes** (*int*) – Length of expected reply from request (if any) (Default: 0)

**Returns** Returns a numpy array of ints if reply expected, otherwise None

**Return type** `numpy.array` of int or None

**Raises** `PymePixException` – Communication error

**requestGetBytes** (*cmd, dev\_nr, expected\_bytes, args=0*)

**requestGetInt** (*cmd, dev\_nr, arg=0*)

**requestGetIntBytes** (*cmd, dev\_nr, expected\_bytes, args=0*)

**requestGetInts** (*cmd, dev\_nr, num\_ints, args=0*)

**requestSetInt** (*cmd, dev\_nr, value*)

**requestSetIntBytes** (*cmd, dev\_nr, value\_int, value\_bytes*)

**requestSetInts** (*cmd, dev\_nr, value*)

**resetCounters** ()

**resetDevices** ()

Resets all devices

**resetModule** (*readout\_speed*)

Resets the SPIDR board and sets a new readout speed

**Parameters** **readout\_speed** (`SpidrReadoutSpeed`) – Read-out speed the device will operate at

## Notes

Its not clear if this does anything as its not usually used

**resetPacketCounters** ()

**resetTimers** ()

Resets all timers to zero

Sets the internal 48-bit timers for all Timepix/Medipix devices to zero

**Raises** `PymePixException` – Communication error

**restartTimers** ()

Restarts SPIDR and Device timers

Synchronizes both the SPIDR clock and Timepix/Medipix clocks so both trigger and ToA timestamps match

---

**Important:** This must be done if event selection is required (e.g. time of flight) otherwise the timestamps will be offset

---

**Raises** `PymePixException` – Communication error

**sequentialReadout** (*tokens, now*)

**setBiasSupplyEnable** (*enable*)

Enables/Disables bias supply voltage

**Parameters** **enable** (*bool*) – True - enables bias supply voltage False - disables bias supply voltage

**Raises** `PymePixException` – Communication error

**setBusy** ()

**setPowerPulseEnable** (*enable*)

**setShutterTriggerConfig** (*mode, length\_us, freq\_hz, count, delay\_ns=0*)

Set the shutter configuration in one go

**Parameters**

- **mode** (*int*) – Shutter trigger mode
- **length\_us** (*int*) – Shutter open time in microseconds
- **freq\_hz** (*int*) – Auto trigger frequency in Hertz
- **count** (*int*) – Number of triggers
- **delay\_ns** (*int, optional*) – Delay between each trigger (Default: 0)

**Raises** `PymePixException` – Communication error

**setSpidrReg** (*addr, value*)

**setTpxPowerPulseEnable** (*enable*)

**shutterCounter**

**shutterTriggerConfig**

**softwareVersion**

Software version

**Returns** Version number of software in the SPIDR board**Return type** int**Raises** `PymePixException` – Communication error**spidrFanSpeed****startAutoTrigger()**

Starts the auto trigger

**Raises** `PymePixException` – Communication error**stopAutoTrigger()**

Stops the auto trigger

**Raises** `PymePixException` – Communication error**vdd****vddNow**`pymepix.SPIDR.spidrcontroller.main()`**pymepix.SPIDR.spidrdefs module**

Module that contains constants that can be passed into spidr

**class** `pymepix.SPIDR.spidrdefs.SpidrReadoutSpeed`Bases: `enum.Enum`

An enumeration.

**Default** = 0**HighSpeed** = 2309737967**LowSpeed** = 305419896**class** `pymepix.SPIDR.spidrdefs.SpidrRegs`Bases: `enum.IntEnum`

An enumeration.

**SPIDR\_CPU2TPX\_WR\_I** = 456**SPIDR\_DEVICES\_AND\_PORTS\_I** = 704**SPIDR\_FE\_GTX\_CTRL\_STAT\_I** = 768**SPIDR\_IPMUX\_CONFIG\_I** = 896**SPIDR\_PIXEL\_FILTER\_I** = 916**SPIDR\_PIXEL\_PKTCounter\_I** = 832**SPIDR\_PIXEL\_PKTCounter\_OLD\_I** = 912**SPIDR\_SHUTTERTRIG\_CNT\_I** = 660**SPIDR\_SHUTTERTRIG\_CTRL\_I** = 656**SPIDR\_SHUTTERTRIG\_DELAY\_I** = 684

```

SPIDR_SHUTTERTRIG_FREQ_I = 664
SPIDR_SHUTTERTRIG_LENGTH_I = 668
SPIDR_TDC_TRIGGERCOUNTER_I = 760
SPIDR_UDPMON_PKTCOUNTER_I = 904
SPIDR_UDPPAUSE_PKTCOUNTER_I = 908
SPIDR_UDP_PKTCOUNTER_I = 900

```

```
class pymepix.SPIDR.spidrdefs.SpidrShutterMode
```

Bases: `enum.Enum`

An enumeration.

```

Auto = 4
ExternalFallingRising = 1
ExternalFallingTimer = 3
ExternalRisingFalling = 0
ExternalRisingTimer = 2
Open = 6
PulseCounter = 5

```

## pymepix.SPIDR.spidrdevice module

```
class pymepix.SPIDR.spidrdevice.SpidrDevice(spidr_ctrl, device_num)
```

Bases: `pymepix.core.log.Logger`

Object that interfaces with a specific device (Timepix/Medipix) connect to SPIDR

This object handles communication and management of a specific device. There is no need to create this object directly as `SpidrController` automatically creates it for you and is accessed by its [] getter methods

### Parameters

- **spidr\_ctrl** (`SpidrController`) – SPIDR controller object the device belongs to
- **device\_num** – Device index from SPIDR (Starts from 1)

**TpPeriodPhase**

**clearPixelConfig()**

**columnTestPulseRegister**

**deviceId**

Returns unique device Id

### Parameters

- **spidr\_ctrl** (`SpidrController`) – SPIDR controller object the device belongs to
- **device\_num** – Device index from SPIDR (Starts from 1)

**devicePort**

**genConfig**

**getDac** (*dac\_code*)

```
getDacOut (nr_samples)
getPixelConfig ()
headerFilter
ipAddrDest
ipAddrSrc
linkStatus
outBlockConfig
pixelPacketCounter
pllConfig
powerPulseConfig
readoutSpeed
reinitDevice ()
reset ()
resetPixelConfig (index=-1, all_pixels=False)
resetPixels ()
serverPort
setDac (dac_code, dac_val)
setDacDefault ()
setExternalDac (dac_code, dac_val)
setHeaderFilter (eth_mask, cpu_mask)
setOutputMask (value)
setPixelMask (mask)
setPixelTestBit (test)
setPixelThreshold (threshold)
setSenseDac (dac_code)
setSinglePixelMask (x, y, mask)
setSinglePixelTestBit (x, y, val)
setSinglePixelThreshold (x, y, threshold)
setTpPeriodPhase (period, phase)
shutterEnd
shutterStart
slaveConfig
t0Sync ()
timer
tpNumber
uploadPacket (packet)
```

```
uploadPixelConfig (formatted=True, columns_per_packet=1)
```

## Module contents

### pymepix.clustering package

#### Submodules

#### pymepix.clustering.cluster\_stream module

```
class pymepix.clustering.cluster_stream.ClusterStream (dim=256, max_dist_tof=1e-08, min_cluster_size=3, tot_offset=0.5, *args, **kwargs)
```

Bases: object

```
perform (data)
```

## Module contents

### pymepix.config package

#### Submodules

#### pymepix.config.defaultconfig module

```
class pymepix.config.defaultconfig.DefaultConfig
Bases: pymepix.config.timepixconfig.TimepixConfig
```

Provides default values for DAC parameters

```
biasVoltage ()
Returns bias Voltage
```

```
dacCodes ()
Accessor for the dac parameters
```

**Returns** The value for every DAC parameter

**Return type** list of tuples (<dac code>, <value>)

```
maskPixels
Returns mask pixels
```

```
testPixels
Returns test pixels
```

```
thresholdPixels
Returns threshold pixels
```

#### pymepix.config.load\_config module

```
pymepix.config.load_config.load_config (config_name='default.yaml')
```

## pymepix.config.sophyconfig module

**class** pymepix.config.sophyconfig.**SophyConfig** (*filename*)  
Bases: [pymepix.config.timepixconfig.TimepixConfig](#), [pymepix.core.log.Logger](#)

This class provides functionality for interpreting a .spx config file from SoPhy.

**biasVoltage** ()  
Returns bias Voltage

**dacCodes** ()  
Accessor for the dac parameters  
**Returns** The value for every DAC parameter  
**Return type** list of tuples (<dac code>, <value>)

**filename**

**loadFile** (*filename*)

**maskPixels**  
Accessor for the mask pixels [0, 1]  
**Returns** The information which pixels are to be masked  
**Return type** `numpy.ndarray (256, 256)`

**parseDAC** (*xmlstring*)  
Reads and formats DAC parameters

**parsePixelConfig** (*zip\_file, file\_names*)  
Reads and formats the pixel data from config file.

### Notes

The spx config file saves the pixel information row by row while the timepix camera expects the information column wise.

**saveMask** ()

**testPixels**  
Accessor for the test pixels

**Returns**  
**Return type** `numpy.ndarray (256, 256)`

**thresholdPixels**  
Accessor for the pixel thresholds [0, 15]

**Returns** The threshold information for each pixel  
**Return type** `numpy.ndarray (256, 256)`

`pymepix.config.sophyconfig.main()`

## pymepix.config.timepixconfig module

**class** pymepix.config.timepixconfig.**TimepixConfig**  
Bases: `abc.ABC`



**biasVoltage()**  
Returns bias Voltage

**dacCodes()**  
Returns an iterator with format daccode,value

**maskPixels**  
Returns mask pixels

**testPixels**  
Returns test pixels

**thresholdPixels**  
Returns threshold pixels

## Module contents

### pymepix.core package

#### Submodules

#### pymepix.core.log module

**class** pymepix.core.log.**Logger** (*name*)  
Bases: pymepix.core.log.PymepixLogger  
Standard logging using logger library

**Parameters** **name** (*str*) – Name used for logging

**getLogger** (*name*)

**class** pymepix.core.log.**ProcessLogger** (*name*)  
Bases: pymepix.core.log.PymepixLogger  
Sends logs to queue to be processed by logging thread

**Parameters** **name** (*str*) – Name used for logging

**getLogger** (*name*)

## Module contents

### pymepix.processing package

#### Subpackages

#### pymepix.processing.logic package

#### Submodules

**pymepix.processing.logic.centroid\_calculator module**

```
class pymepix.processing.logic.centroid_calculator.CentroidCalculator (cent_timewalk_lut=None,  
                                                                    num-  
                                                                    ber_of_processes=4,  
                                                                    clus-  
                                                                    ter-  
                                                                    ing_args={},  
                                                                    db-  
                                                                    scan_clustering=True,  
                                                                    *args,  
                                                                    **kwargs)
```

Bases: `pymepix.processing.logic.processing_step.ProcessingStep`

Class responsible for calculating centroids in timepix data. This includes the calculation of the clusters first and the centroids. The data processed is not the direct raw data but the data that has been processed by the PacketProcessor before (x, y, tof, tot).

**process(data) :**

Process data and return the result. To use this class only this method should be used! Use the other methods only for testing or if you are sure about what you are doing

**calculate\_centroids\_cluster\_stream(chunk)****calculate\_centroids\_dbscan(chunk)****calculate\_centroids\_properties(shot, x, y, tof, tot, labels)**

Calculates the properties of the centroids from labeled data points.

ATTENTION! The order of the points can have an impact on the result due to errors in the floating point arithmetics.

Very simple example: `arr = np.random.random(100)` `arr.sum() - np.sort(arr).sum()` This example shows that there is a very small difference between the two sums. The inaccuracy of floating point arithmetics can depend on the order of the values. Strongly simplified  $(3.2 + 3.4) + 2.7$  and  $3.2 + (3.4 + 2.7)$  can be unequal for floating point numbers.

Therefore there is no guarantee for strictly equal results. Even after sorting. The error we observed can be about  $10^{-22}$  nano seconds.

Currently this issue exists only for the TOF-column as the other columns are integer-based values.

**centroid\_chunks\_to\_centroids(chunks)**

`centroids = [[] for i in range(7)]` for chunk in list(chunks):

**if chunk != None:**

**for index, coordinate in enumerate(chunk):** centroids[index].append(coordinate)

**cluster\_stream\_preprocess(shot, x, y, tof, tot)****cs\_max\_dist\_tof**

Setting the maximal ToF distance between the voxels belonging to the cluster in Cluster Streaming algorithm

**cs\_min\_cluster\_size**

Setting the minimal cluster size in Cluster Streaming algorithm

**cs\_sensor\_size**

Setting for the number of packets skipped during processing. Every packet\_skip packet is processed. This means for a value of 1 every packet is processed. For 2 only every 2nd packet is processed.

**cs\_tot\_offset**

Setting the ToT ratio factor of the voxel to the ToT of previous voxel in Cluster Streaming algorithm. Zero factor means ToT of prev. voxel should be larger. 0.5 factor means ToT of prev voxel could be high than the half of the considered voxel

**dbscan\_clustering****epsilon****min\_samples****perform\_centroiding\_cluster\_stream** (*chunks*)**perform\_centroiding\_dbscan** (*chunks*)**perform\_clustering\_dbscan** (*shot, x, y, tof*)

The clustering with DBSCAN, which is performed in this function is dependent on the order of the data in rare cases. Therefore, reordering in any means can lead to slightly changed results, which should not be an issue.

Martin Ester, Hans-Peter Kriegel, Jiirg Sander, Xiaowei Xu: A Density Based Algorithm for Discovering Clusters [p. 229-230] (<https://www.aaii.org/Papers/KDD/1996/KDD96-037.pdf>) A more specific explanation can be found here: <https://stats.stackexchange.com/questions/306829/why-is-dbscan-deterministic>

**process** (*data*)**tot\_threshold**

Determines which time over threshold values to filter before centroiding

This is useful in reducing the computational time in centroiding and can filter out noise.

**triggers\_processed**

Setting for the number of packets skipped during processing. Every packet\_skip packet is processed. This means for a value of 1 every packet is processed. For 2 only every 2nd packet is processed.

**class** pymepix.processing.logic.centroid\_calculator.**CentroidCalculatorPooled** (*number\_of\_processes*,  
\*args,  
\*\*kwargs)

Bases: `pymepix.processing.logic.centroid_calculator.CentroidCalculator`

Parallelized implementation of CentroidCalculator using mp.Pool for parallelization.

**perform\_centroiding** (*chunks*)**post\_process** ()**pre\_process** ()

`pymepix.processing.logic.centroid_calculator.calculate_centroids_dbscan` (*chunk*,  
*tot\_threshold*,  
*\_tof\_scale*,  
*ep-*  
*silon*,  
*min\_samples*,  
*\_cent\_timewalk\_lut*)

```
pymepix.processing.logic.centroid_calculator.calculate_centroids_properties (shot,
                                                                              x,
                                                                              y,
                                                                              tof,
                                                                              tot,
                                                                              la-
                                                                              bels,
                                                                              _cent_timewalk_lut)
```

Calculates the properties of the centroids from labeled data points.

ATTENTION! The order of the points can have an impact on the result due to errors in the floating point arithmetics.

Very simple example: `arr = np.random.random(100)` `arr.sum() - np.sort(arr).sum()` This example shows that there is a very small difference between the two sums. The inaccuracy of floating point arithmetics can depend on the order of the values. Strongly simplified  $(3.2 + 3.4) + 2.7$  and  $3.2 + (3.4 + 2.7)$  can be unequal for floating point numbers.

Therefore there is no guarantee for strictly equal results. Even after sorting. The error we observed can be about  $10^{-22}$  nano seconds.

Currently this issue exists only for the TOF-column as the other columns are integer-based values.

```
pymepix.processing.logic.centroid_calculator.perform_clustering_dbscan (shot,
                                                                           x, y,
                                                                           tof,
                                                                           _tof_scale,
                                                                           ep-
                                                                           silon,
                                                                           min_samples)
```

The clustering with DBSCAN, which is performed in this function is dependent on the order of the data in rare cases. Therefore, reordering in any means can lead to slightly changed results, which should not be an issue.

Martin Ester, Hans-Peter Kriegel, Jiirg Sander, Xiaowei Xu: A Density Based Algorithm for Discovering Clusters [p. 229-230] (<https://www.aai.org/Papers/KDD/1996/KDD96-037.pdf>) A more specific explanation can be found here: <https://stats.stackexchange.com/questions/306829/why-is-dbscan-deterministic>

## pymepix.processing.logic.packet\_processor module

```
class pymepix.processing.logic.packet_processor.PacketProcessor (handle_events=True,
                                                                  event_window=(0.0,
                                                                  10000.0), posi-
                                                                  tion_offset=(0,
                                                                  0), orienta-
                                                                  tion=<PixelOrientation.Up:
                                                                  0>,
                                                                  start_time=0,
                                                                  time-
                                                                  walk_lut=None,
                                                                  *args,
                                                                  **kwargs)
```

Bases: `pymepix.processing.logic.processing_step.ProcessingStep`

Class responsible to transform the raw data coming from the timepix directly into an easier processible data format. Takes into account the pixel- and trigger data to calculate toa and tof dimensions.

**process (data) :**

Process data and return the result. To use this class only this method should be used! Use the other methods

only for testing or if you are sure about what you are doing

**clearBuffers** ()

**correct\_global\_time** (*arr, ltime*)

**event\_window**

**find\_events\_fast** ()

**find\_events\_fast\_post** ()

Call this function at the very end of to also have the last two trigger events processed

**getBuffers** (*val\_filter=None*)

**handle\_events**

**Type** noindex

**orientPixels** (*col, row*)

Orient the pixels based on Timepix orientation

**post\_process** ()

**pre\_process** ()

**process** (*data*)

**process\_pixels** (*pixdata, longtime*)

**process\_trigger1** (*pixdata, longtime*)

**process\_trigger2** (*tidtrigdata, longtime*)

**updateBuffers** (*val\_filter*)

**class** pymepix.processing.logic.packet\_processor.PixelOrientation

Bases: `enum.IntEnum`

Defines how row and col are interpreted in the output

**Down** = 2

x=-column, y = -row

**Left** = 1

x=row, y=-column

**Right** = 3

x=-row, y=column

**Up** = 0

Up is the default, x=column,y=row

## pymepix.processing.logic.processing\_step module

**class** pymepix.processing.logic.processing\_step.ProcessingStep (*name, param-*

*ter\_wrapper\_class=<class*

*'pymepix.processing.logic.shared\_process*

Bases: `pymepix.core.log.Logger`, `abc.ABC`

Representation of one processing step in the pipeline for processing timepix raw data. Implementations are provided by PacketProcessor and CentroidCalculator. To combine those (and possibly other) classes into a pipeline they have to implement this interface. Also provides pre- and post-process implementations which are required for integration in the online processing pipeline (see PipelineCentroidCalculator and PipelinePacketProcessor).

Currently the picture is the following:

- For post processing the CentroidCalculator and the PacketProcessor are used directly
- PipelineCentroidCalculator and PipelinePacketProcessor build on top of CentroidCalculator and PacketProcessor to provide an integration in the existing online processing pipeline for online analysis.

```
post_process ()
```

```
pre_process ()
```

```
process (data)
```

## pymepix.processing.logic.shared\_processing\_parameter module

```
class pymepix.processing.logic.shared_processing_parameter.SharedProcessingParameter (value)
    Bases: object
```

Variang of the ProcessingParameter used for sharing among multiple processes. This class has to be used if running with the multiprocessing pipeline to ensure all instances of the processing classes are updated when parameters are changed.

```
value
```

```
exception pymepix.processing.logic.shared_processing_parameter.UnknownParameterTypeException
    Bases: Exception
```

## Module contents

### Submodules

## pymepix.processing.acquisition module

Module that contains predefined acquisition pipelines for the user to use

```
class pymepix.processing.acquisition.CentroidPipeline (data_queue, address, long-
                                                         time)
    Bases: pymepix.processing.acquisition.PixelPipeline
```

A Pixel pipeline that includes centroiding

Same as the pixel pipeline but also includes centroid processing, note that this can be extremely slow when dealing with a huge number of objects

```
numBlobProcesses
```

Number of python processes to spawn for centroiding

Setting this will spawn the appropriate number of processes to perform centroiding. Changes take effect on next acquisition.

```
class pymepix.processing.acquisition.PixelPipeline (data_queue, address, longtime,
                                                         use_event=False, name='Pixel',
                                                         event_window=(0,0.001))
    Bases: pymepix.processing.baseacquisition.AcquisitionPipeline
```

An acquisition pipeline that includes the udpsampler and pixel processor

A pipeline that will read from a UDP address and decode the pixels a useable form. This class can be used as a base for all acquisition pipelines.

## pymepix.processing.baseacquisition module

Module deals with managing processing objects to form a data pipeline

**class** pymepix.processing.baseacquisition.**AcquisitionPipeline** (*name*,  
*data\_queue*)

Bases: *pymepix.core.log.Logger*

Class that manages various stages

**addStage** (*stage\_number*, *pipeline\_class*, \**args*, *num\_processes=1*, \*\**kwargs*)  
Adds a stage to the pipeline

**getStage** (*stage\_number*)

**isRunning**

**stages**

**start** ()  
Starts all stages

**stop** ()  
Stops all stages

**class** pymepix.processing.baseacquisition.**AcquisitionStage** (*stage*,  
*num\_processes=1*)

Bases: *pymepix.core.log.Logger*

Defines a single acquisition stage

Usually not created directly. Instead created by *AcquisitionPipeline* Represent a single pipeline stage and handles management of queues and message passing as well as creation and destruction of processing objects.

Processes are not created until build() is called and do not run until start() is called

**Parameters** **stage** (*int*) – Initial position in the pipeline, lower stages are executed first

**build** (*input\_queue=None*, *output\_queue=None*, *file\_writer=None*)

**configureStage** (*pipeline\_class*, \**args*, \*\**kwargs*)  
Configures the stage with a particular processing class

### Parameters

- **pipeline\_class** (*BasePipeline*) – A pipeline class object
- **\*args** – positional arguments to pass into the class init
- **\*\*kwargs** – keyword arguments to pass into the class init

### numProcess

Number of processes to spawn when built

**Parameters** **value** (*int*) – Number of processes to spawn when acquisition starts

**Returns** Number of processes

**Return type** *int*

### outputQueue

### processes

**setArgs** (\**args*, \*\**kwargs*)

**stage**

Current position in the pipeline

**start()**

**startTrainID()**

**stop** (*force=False*)

**stopTrainID()**

`pymepix.processing.baseacquisition.main()`

## pymepix.processing.basepipeline module

Base implementation of objects relating to the processing pipeline

```
class pymepix.processing.basepipeline.BasePipelineObject (name, in-  
                                                         put_queue=None,  
                                                         create_output=True,  
                                                         num_outputs=1,  
                                                         shared_output=None,  
                                                         propagate_input=True)
```

Bases: `multiprocessing.context.Process`, `pymepix.core.log.ProcessLogger`

Base class for integration in a processing pipeline

### Parameters

- **name** (*str*) – Name used for logging
- **input\_queue** (`multiprocessing.Queue`, optional) – Data queue to perform work on (usually) from previous step in processing pipeline
- **create\_output** (*bool, optional*) – Whether this creates its own output queue to pass data, ignored if (Default: True)
- **num\_outputs** (*int, optional*) – Used with `create_output`, number of output queues to create (Default: 1)
- **shared\_output** (`multiprocessing.Queue`, optional) – Data queue to pass results into, useful when multiple processes can put data into the same queue (such as results from centroiding). Ignored if `create_output` is True (Default: None)
- **propagate\_input** (*bool*) – Whether the input data should be propagated further down the chain

### enable

Enables processing

Determines whether the class will perform processing, this has the result of signalling the process to terminate. If there are objects ahead of it then they will stop receiving data if an input queue is required then it will get from the queue before checking processing This is done to prevent the queue from growing when a process behind it is still working

**Parameters** **value** (*bool*) – Enable value

**Returns** Whether the process is enabled or not

**Return type** `bool`



**classmethod hasOutput ()**

Defines whether this class can output results or not, e.g. Centroiding can output results but file writing classes do not

**Returns** Whether results are generated

**Return type** bool

**outputQueues**

Exposes the outputs so they may be connected to the next step

**Returns** All of the outputs

**Return type** list of multiprocessing.Queue

**post\_run ()**

Function called after main processing loop, override to

**pre\_run ()**

Function called before main processing loop, override to

**process (data\_type=None, data=None)**

Main processing function, override this to perform work

To perform work within the pipeline, a class must override this function. General guidelines include, check for correct data type, and must return None for both if no output is given.

**pushOutput (data\_type, data)**

Pushes results to output queue (if available)

**Parameters**

- **data\_type** (*int*) – Identifier for data type (see MessageType for types)
- **data** (*any*) – Results from processing (must be picklable)

**run ()**

Method to be run in sub-process; can be overridden in sub-class

pymepix.processing.basepipeline.main ()

**pymepix.processing.datatypes module**

Defines data that is passed between processing objects

**class pymepix.processing.datatypes.MessageType**

Bases: enum.IntEnum

Defines the type of message that is being passed into a multiprocessing queue

**CentroidData = 3**

Centroided Data

**CloseFileCommand = 5**

Close File Message

**EventData = 2**

Event Data

**OpenFileCommand = 4**

Open File message

**PixelData = 1**

Decoded Pixel/Trigger Data

```

RawData = 0
    Raw UDP packets

TriggerData = 8
    Decoded Triggers

```

## pymepix.processing.pipeline\_centroid\_calculator module

Processors relating to centroiding

```

class pymepix.processing.pipeline_centroid_calculator.PipelineCentroidCalculator (centroid_calculator: pymepix.processing.pipeline_centroid_calculator.PipelineCentroidCalculator,
    =
    <pymepix.processing.pipeline_centroid_calculator.PipelineCentroidCalculator object>,
    in-
    put_queue=None,
    create_output=True,
    num_outputs=1,
    shared_output=None)

```

Bases: *pymepix.processing.basepipeline.BasePipelineObject*

Performs centroiding on EventData recieved from Packet processor

**process** (*data\_type=None, data=None*)

Main processing function, override this do perform work

To perform work within the pipeline, a class must override this function. General guidelines include, check for correct data type, and must return None for both if no output is given.

## pymepix.processing.pipeline\_packet\_processor module

```

class pymepix.processing.pipeline_packet_processor.PipelinePacketProcessor (packet_processor: pymepix.processing.pipeline_packet_processor.PipelinePacketProcessor,
    =
    <pymepix.processing.pipeline_packet_processor.PipelinePacketProcessor object>,
    in-
    put_queue=None,
    create_output=True,
    num_outputs=1,
    shared_output=None)

```

Bases: *pymepix.processing.basepipeline.BasePipelineObject*

Processes Pixel packets for ToA, ToT, triggers and events

This class, creates a UDP socket connection to SPIDR and recieves the UDP packets from Timepix It then pre-processes them and sends them off for more processing

**init\_new\_process** ()

**post\_run** ()

Function called after main processing loop, override to

**pre\_run()**

Function called before main processing loop, override to

**process** (*data\_type=None, data=None*)

Main processing function, override this do perform work

To perform work within the pipeline, a class must override this function. General guidelines include, check for correct data type, and must return None for both if no output is given.

## pymepix.processing.rawfilesampler module

```
class pymepix.processing.rawfilesampler.RawFileSampler(file_name, output_file, number_of_processes=None, timewalk_file=None, cent_timewalk_file=None, progress_callback=None, clustering_args={}, db_scan_clustering=True, **kwargs)
```

Bases: object

**bytes\_from\_file** (*chunksize=8192*)

**handle\_lsb\_time** (*pixdata*)

**handle\_msb\_time** (*pixdata*)

**handle\_other** (*pixdata*)

trash data which arrives before 1st timestamp data (heartbeat)

**init\_new\_process** (*file*)

create connections and initialize variables in new process

**post\_run()**

**pre\_run()**

init stuff which should only be available in new process

**push\_data** (*post=False*)

**run()**

method which is executed in new process via multiprocessing.Process.start

**saveToHDF5** (*output\_file, raw, clusters, timeStamps, trigger1, trigger2*)

## pymepix.processing.rawtodisk module

```
class pymepix.processing.rawtodisk.Raw2Disk(context=None)
```

Bases: [pymepix.core.log.ProcessLogger](#)

Class for asynchronously writing raw files Intended to allow writing of raw data while minimizing impact on UDP reception reliability.

**close** (*socket*)

Close the file currently in progress. call in main below

**open\_file** (*socket, filename*)

Creates a file with a given filename and path.

this doesn't work anylonger using 2 sockets for the communication functionality needs to be put outside where you have access to the socket

**write** (*data*)

Writes data to the file. Parameter is buffer type (e.g. bytearray or memoryview)

Not sure how useful this function actually is... It completes the interface for this class but from a performance point of view it doesn't improve things. How could this be benchmarked?

`pymepix.processing.rawtodisk.main()`

`pymepix.processing.rawtodisk.main_process()`

seperate process not strictly necessary, just to double check if this also works with multiprocessing doesn't work for debugging

## pymepix.processing.udpsampler module

**class** `pymepix.processing.udpsampler.UdpSampler` (*address, longtime, chunk\_size=10000, flush\_timeout=0.3, input\_queue=None, create\_output=True, num\_outputs=1, shared\_output=None*)

Bases: `multiprocessing.context.Process`, `pymepix.core.log.ProcessLogger`

Recieves udp packets from SPDIR

This class, creates a UDP socket connection to SPIDR and recieves the UDP packets from Timepix It them pre-processes them and sends them off for more processing

**close\_file**

**create\_socket\_connection** (*address*)

Establishes a UDP connection to spidr

**enable**

Enables processing

Determines whether the class will perform processing, this has the result of signalling the process to terminate. If there are objects ahead of it then they will stop receiving data if an input queue is required then it will get from the queue before checking processing This is done to prevent the queue from growing when a process behind it is still working

**Parameters** *value* (*bool*) – Enable value

**Returns** Whether the process is enabled or not

**Return type** `bool`

**get\_useful\_packets** (*packet*)

**init\_new\_process** ()

create connections and initialize variables in new process

**outfile\_name**

**post\_run** ()

method get's called either at the very end of the process live or if there's a socket timeout and raw2disk file should be closed

**pre\_run** ()

init stuff which should only be available in new process

**record**

Enables saving data to disk

Determines whether the class will perform processing, this has the result of signalling the process to terminate. If there are objects ahead of it then they will stop receiving data if an input queue is required then it will get from the queue before checking processing This is done to prevent the queue from growing when a process behind it is still working

**Parameters** **value** (*bool*) – Enable value

**Returns** Whether the process should record and write to disk or not

**Return type** bool

**run()**

method which is executed in new process via multiprocessing.Process.start

`pymepix.processing.udpsampler.main()`

**pymepix.processing.usbtrainid module**

**class** `pymepix.processing.usbtrainid.USBTrainID` (*name='USBTrainId'*)

Bases: `multiprocessing.context.Process`, `pymepix.core.log.ProcessLogger`

Class for asynchronously writing raw files Intended to allow writing of raw data while minimizing impact on UDP reception reliability

**connect\_device** (*device*)

Establish connection to USB device

**run()**

Method to be run in sub-process; can be overridden in sub-class

`pymepix.processing.usbtrainid.main()`

**Module contents****pymepix.util package****Submodules****pymepix.util.spidrDummyTCP module**

**class** `pymepix.util.spidrDummyTCP.TPX3Handler` (*request, client\_address, server*)

Bases: `socketserver.BaseRequestHandler`, `pymepix.core.log.Logger`

**handle()**

`pymepix.util.spidrDummyTCP.main()`

**pymepix.util.spidrDummyUDP module**

`pymepix.util.spidrDummyUDP.main()`

## pymepix.util.storage module

Useful functions to store data

`pymepix.util.storage.open_output_file(filename, ext, index=0)`

`pymepix.util.storage.store_centroid(f, data)`

`pymepix.util.storage.store_raw(f, data)`

`pymepix.util.storage.store_toa(f, data)`

`pymepix.util.storage.store_tof(f, data)`

## pymepix.util.tcpsampler module

```
class pymepix.util.tcpsampler.TcpSampler(address,      longtime,      chunk_size=10000,
                                          flush_timeout=0.3,  input_queue=None,
                                          create_output=True,   num_outputs=1,
                                          shared_output=None)
```

Bases: `multiprocessing.context.Process`, `pymepix.core.log.ProcessLogger`

Receives tcp packets

The same as UdpSampler just with TCP

**close\_file**

**createConnection** (*address*)

Establishes a TCP connection

**enable**

Enables processing

Determines whether the class will perform processing, this has the result of signalling the process to terminate. If there are objects ahead of it then they will stop receiving data if an input queue is required then it will get from the queue before checking processing This is done to prevent the queue from growing when a process behind it is still working

**Parameters** *value* (*bool*) – Enable value

**Returns** Whether the process is enabled or not

**Return type** *bool*

**get\_useful\_packets** (*packet*)

**init\_new\_process** ()

**outfile\_name**

**post\_run** ()

**pre\_run** ()

**record**

Enables saving data to disk

Determines whether the class will perform processing, this has the result of signalling the process to terminate. If there are objects ahead of it then they will stop receiving data if an input queue is required then it will get from the queue before checking processing This is done to prevent the queue from growing when a process behind it is still working

**Parameters** *value* (*bool*) – Enable value

**Returns** Whether the process should record and write to disk or not

**Return type** bool

**run()**

Method to be run in sub-process; can be overridden in sub-class

**stopRaw2Disk()**

```
self.debug('Stopping Raw2Disk') self.write2disk.close() self.write2disk.my_sock.send_string('SHUTDOWN')
# print(write2disk.my_sock.recv()) self.write2disk.write_thr.join() self.debug('Raw2Disk stopped')
```

`pymepix.util.tcpsampler.main()`

## pymepix.util.timewalk module

`pymepix.util.timewalk.compute_timewalk(tof, tot, region)`

`pymepix.util.timewalk.compute_timewalk_lookup(tof, tot, region)`

## Module contents

### Submodules

#### pymepix.main module

Main module for pymepix

`pymepix.main.connect_timepix(args)`

`pymepix.main.main()`

`pymepix.main.post_process(args)`

#### pymepix.post\_processing module

```
class pymepix.post_processing.ProgressBar(iterable=None, desc=None, total=None,
leave=True, file=None, ncols=None, mininterval=0.1, maxinterval=10.0, miniters=None,
ascii=None, disable=False, unit='it',
unit_scale=False, dynamic_ncols=False,
smoothing=0.3, bar_format=None, initial=0, position=None, postfix=None,
unit_divisor=1000, write_bytes=False,
lock_args=None, nrows=None, colour=None,
delay=0, gui=False, **kwargs)
```

Bases: `tqdm.std.tqdm`

**gui\_bar\_fun** = None

**update\_to**(progress)

```
pymepix.post_processing.run_post_processing(input_file_name,          output_file,
                                           number_processes,         time-
                                           walk_file,                cent_timewalk_file,
                                           progress_callback=<function updatePro-
                                           gressBar>, clustering_args={}, db-
                                           scan_clustering=True, **kwargs)

pymepix.post_processing.updateProgressBar(progress)
```

## pymepix.pymepix\_connection module

**exception** pymepix.pymepix\_connection.PollBufferEmpty

Bases: Exception

```
class pymepix.pymepix_connection.PymepixConnection(spidr_address=('192.168.1.10',
                                                                    50000),
                                                    src_ip_port=('192.168.1.1',
                                                                8192),
                                                    pipeline_class=<class
                                                                'pymepix.processing.acquisition.PixelPipeline'>)
```

Bases: [pymepix.core.log.Logger](#)

High level class to work with timepix and perform acquisition

This class performs connection to SPIDR, initialization of timepix and handling of acquisition. Each individual timepix device can be accessed using the square bracket operator.

### Parameters

- **spidr\_address** (tuple of str and int) – socket style tuple of SPIDR ip address and port
- **src\_ip\_port** (tuple of str and int, optional) – socket style tuple of the IP address and port of the interface that is connecting to SPIDR

## Examples

Startup device

```
>>> timepix = Pymepix(('192.168.1.10', 50000))
```

Find how many Timepix are connected

```
>>> len(timepix)
1
```

Set the Bias voltage >>> timepix.biasVoltage = 50

Access a specific Timepix device:

```
>>> timepix[0].deviceName
W0026_K06
```

Load a config file into timepix

```
>>> timepix[0].loadSophyConfig('W0026_K06_50V.spx')
```

### biasVoltage

Bias voltage in volts



**dataCallback**

Function to call when data is received from a timepix device

This has the effect of disabling polling.

**data\_thread()****enablePolling** (*maxlen=100*)

Enables polling mode

This clears any user defined callbacks and the polling buffer

**getDevice** (*num*) → pymepix.timepixdevice.TimepixDevice

**isAcquiring****numDevices****poll** (*block=False*)

If polling is used, returns data stored in data buffer.

the buffer is in the form of a ring and will overwrite older values if it becomes full

**Returns**

**Return type** MessageType , data

**pollBufferLength**

Get/Set polling buffer length

Clears buffer on set

**start** ()

Starts acquisition

**stop** ()

Stops acquisition

**pymepix.timepixdef module****class** pymepix.timepixdef.DacRegisterCodes

Bases: enum.IntEnum

An enumeration.

**Ibias\_CP\_PLL** = 17

**Ibias\_DiscS1\_OFF** = 9

**Ibias\_DiscS1\_ON** = 8

**Ibias\_DiscS2\_OFF** = 11

**Ibias\_DiscS2\_ON** = 10

**Ibias\_Ikrum** = 4

**Ibias\_PixelDAC** = 12

**Ibias\_Preamp\_OFF** = 2

**Ibias\_Preamp\_ON** = 1

**Ibias\_TPbufferIn** = 13

**Ibias\_TPbufferOut** = 14

```
    PLL_Vcntrl = 18
    VPreamp_NCAS = 3
    VTP_coarse = 15
    VTP_fine = 16
    Vfbk = 5
    Vthreshold_coarse = 7
    Vthreshold_fine = 6

class pymepix.timepixdef.GrayCounter
    Bases: enum.IntEnum

    An enumeration.

    Disable = 0
    Enable = 8
    Mask = 8

class pymepix.timepixdef.OperationMode
    Bases: enum.IntEnum

    An enumeration.

    EventiTot = 4
    Mask = 6
    ToA = 2
    ToAandToT = 0

class pymepix.timepixdef.PacketType
    Bases: enum.Enum

    An enumeration.

    Pixel = 1
    Trigger = 0

class pymepix.timepixdef.Polarity
    Bases: enum.IntEnum

    An enumeration.

    Negative = 1
    Positive = 0

class pymepix.timepixdef.SuperPixel
    Bases: enum.IntEnum

    An enumeration.

    Disable = 0
    Enable = 64
    Mask = 64
```

```
class pymepix.timepixdef.TestPulse
    Bases: enum.IntEnum

    An enumeration.

    Disable = 0
    Enable = 32
    Mask = 32

class pymepix.timepixdef.TestPulseDigAnalog
    Bases: enum.IntEnum

    An enumeration.

    DiscriminatorDigital = 512
    FrontEndAnalog = 0
    Mask = 512

class pymepix.timepixdef.TestPulseGenerator
    Bases: enum.IntEnum

    An enumeration.

    External = 1024
    Internal = 0
    Mask = 1024

class pymepix.timepixdef.TimeofArrivalClock
    Bases: enum.IntEnum

    An enumeration.

    Mask = 2048
    PhaseShiftedGray = 0
    SystemClock = 2048

class pymepix.timepixdef.TimerOverflow
    Bases: enum.IntEnum

    An enumeration.

    CycleOverflow = 0
    Mask = 128
    StopOverflow = 128
```

### **pymepix.timepixdevice module**

```
exception pymepix.timepixdevice.BadPixelFormat
    Bases: Exception

exception pymepix.timepixdevice.ConfigClassException
    Bases: Exception
```

```
class pymepix.timepixdevice.TimepixDevice (spidr_device, data_queue,  
                                           pipeline_class=<class  
                                           'pymepix.processing.acquisition.PixelPipeline'>)
```

Bases: `pymepix.core.log.Logger`

Provides high level control of a timepix/medipix object

**Ibias\_DiscS1\_OFF**  
[0, 15]

**Ibias\_DiscS1\_ON**  
[0, 255]

**Ibias\_DiscS2\_OFF**  
[0, 15]

**Ibias\_DiscS2\_ON**  
[0, 255]

**Ibias\_Ikrum**  
[0, 255]

**Ibias\_PixelDAC**  
[0, 255]

**Ibias\_Preamp\_OFF**  
[0, 15]

**Ibias\_Preamp\_ON**  
[0, 255]

**Ibias\_TPbufferIn**  
[0, 255]

**Ibias\_TPbufferOut**  
[0, 255]

**VPreamp\_NCAS**  
[0, 255]

**VTP\_coarse**  
[0, 255]

**VTP\_fine**  
[0, 511]

**Vfbk**  
[0, 255]

**Vthreshold\_coarse**  
[0, 15]

**Vthreshold\_fine**  
[0, 511]

**acquisition**

Returns the acquisition object

Can be used to set parameters in the acquisition directly for example, to setup TOF calculation when using a PixelPipeline

```
>>> tpx.acquisition.enableEvents  
False  
>>> tpx.acquistion.enableEvents = True
```

**config**

**devIdToString()**

Converts device ID into readable string

**Returns** Device string identifier

**Return type** str

**deviceName**

**grayCounter**

**loadConfig(\*args, \*\*kwargs)**

Loads dac settings from the Config class

**operationMode**

**pauseHeartbeat()**

**pixelMask**

Pixel mask set for timepix device

**Parameters** **value** (numpy.array of int) – 256x256 uint8 threshold mask to set locally

**Returns** Locally stored pixel mask matrix

**Return type** numpy.array of int or None

**pixelTest**

Pixel test set for timepix device

**Parameters** **value** (numpy.array of int) – 256x256 uint8 pixel test to set locally

**Returns** Locally stored pixel test matrix

**Return type** numpy.array of int or None

**pixelThreshold**

Threshold set for timepix device

**Parameters** **value** (numpy.array of int) – 256x256 uint8 threshold to set locally

**Returns** Locally stored threshold matrix

**Return type** numpy.array of int or None

**polarity**

**refreshPixels()**

Loads timepix pixel configuration to local array

**resetPixels()**

Clears pixel configuration

**resumeHeartbeat()**

**setConfigClass** (klass: *pymepix.config.timepixconfig.TimepixConfig*)

**setDac** (code, value)

Sets the DAC parameter using codes

**Parameters**

- **code** (int) – DAC code to set
- **value** (int) – value to set

**setEthernetFilter** (*eth\_filter*)

Sets the packet filter, usually set to 0xFFFF to all all packets

**setupAcquisition** (*acquisition\_klass*, \*args, \*\*kwargs)

**setupDevice** ()

Sets up valid paramters for acquisition

This will be manual when other acquisition parameters are working

**start** ()

**start\_recording** (*path*)

**stop** ()

**stop\_recording** ()

**superPixel**

**testPulse**

**testPulseDigitalAnalog**

**testPulseGeneratorSource**

**timeOfArrivalClock**

**timerOverflowControl**

**update\_timer** ()

Heartbeat thread

**uploadPixels** ()

Uploads local pixel configuration to timepix

`pymepix.timepixdevice.main()`

## Module contents

## CHAPTER 11

---

### References

---





## CHAPTER 12

---

### Pymepix Documentation

---

[Pymepix](#) is a python library for interfacing, controlling and acquiring from SPIDR-Timepix detectors.

See also the accompanying [Pymepix-viewer](#) for an example user tool.

See [[AlRefaie2019](#)] for a description of version 1.0 of both tools and as a formal, citeable reference and the [online manual](#) for further information on later versions.



---

## Bibliography

---

- [AlRefaie2019] A. Al-Refaie, M. Johny, J. Correa, D. Pennicard, P. Svihra, A. Nomerotski, S. Trippel, J. Küpper: PymePix: a python library for SPIDR readout of Timepix3, *Journal of Instrumentation* **14**, P10003–P10003 (2019), DOI: [10.1088/1748-0221/14/10/p10003](https://doi.org/10.1088/1748-0221/14/10/p10003); arXiv:1905.07999 [physics]



### p

- `pymepix`, 66
- `pymepix.clustering`, 43
- `pymepix.clustering.cluster_stream`, 43
- `pymepix.config`, 45
- `pymepix.config.defaultconfig`, 43
- `pymepix.config.load_config`, 43
- `pymepix.config.sophyconfig`, 44
- `pymepix.config.timepixconfig`, 44
- `pymepix.core`, 45
- `pymepix.core.log`, 45
- `pymepix.main`, 59
- `pymepix.post_processing`, 59
- `pymepix.processing`, 57
- `pymepix.processing.acquisition`, 50
- `pymepix.processing.baseacquisition`, 51
- `pymepix.processing.basepipeline`, 52
- `pymepix.processing.datatypes`, 53
- `pymepix.processing.logic`, 50
- `pymepix.processing.logic.centroid_calculator`, 46
- `pymepix.processing.logic.packet_processor`, 48
- `pymepix.processing.logic.processing_step`, 49
- `pymepix.processing.logic.shared_processing_parameter`, 50
- `pymepix.processing.pipeline_centroid_calculator`, 54
- `pymepix.processing.pipeline_packet_processor`, 54
- `pymepix.processing.rawfilesampler`, 55
- `pymepix.processing.rawtodisk`, 55
- `pymepix.processing.udpsampler`, 56
- `pymepix.processing.usbtrainid`, 57
- `pymepix.pymepix_connection`, 60
- `pymepix.SPIDR`, 43
- `pymepix.SPIDR.error`, 29
- `pymepix.SPIDR.spidrcmds`, 29
- `pymepix.SPIDR.spidrcontroller`, 33
- `pymepix.SPIDR.spidrdefs`, 40
- `pymepix.SPIDR.spidrdevice`, 41
- `pymepix.timepixdef`, 61
- `pymepix.timepixdevice`, 63
- `pymepix.util`, 59
- `pymepix.util.spidrDummyTCP`, 57
- `pymepix.util.spidrDummyUDP`, 57
- `pymepix.util.storage`, 58
- `pymepix.util.tcpsampler`, 58
- `pymepix.util.timewalk`, 59



## A

acquisition (*pymepix.timepixdevice.TimepixDevice* attribute), 64

AcquisitionPipeline (class in *pymepix.processing.baseacquisition*), 51

AcquisitionStage (class in *pymepix.processing.baseacquisition*), 51

addStage() (*pymepix.processing.baseacquisition.AcquisitionPipeline* method), 51

Auto (*pymepix.SPIDR.spidrdefs.SpidrShutterMode* attribute), 41

avdd (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 35

avddNow (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 35

calculate\_centroids\_dbscan() (in module *pymepix.processing.logic.centroid\_calculator*), 47

calculate\_centroids\_dbscan() (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* method), 46

calculate\_centroids\_properties() (in module *pymepix.processing.logic.centroid\_calculator*), 47

calculate\_centroids\_properties() (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* method), 46

centroid\_chunks\_to\_centroids() (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* method), 46

## B

BadPixelFormat, 63

BasePipelineObject (class in *pymepix.processing.basepipeline*), 52

biasVoltage (*pymepix.pymepix\_connection.PymepixConnection* attribute), 60

biasVoltage (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 35

biasVoltage() (*pymepix.config.defaultconfig.DefaultConfig* method), 43

biasVoltage() (*pymepix.config.sophyconfig.SophyConfig* method), 44

biasVoltage() (*pymepix.config.timepixconfig.TimepixConfig* method), 44

build() (*pymepix.processing.baseacquisition.AcquisitionStage* method), 51

bytes\_from\_file() (*pymepix.processing.rawfilesampler.RawFileSampler* method), 55

CentroidCalculator (class in *pymepix.processing.logic.centroid\_calculator*), 46

CentroidCalculatorPooled (class in *pymepix.processing.logic.centroid\_calculator*), 47

CentroidData (*pymepix.processing.datatypes.MessageType* attribute), 53

CentroidPipeline (class in *pymepix.processing.acquisition*), 50

chipboardFanSpeed (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 35

chipboardId (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 35

clearBuffers() (*pymepix.processing.logic.packet\_processor.PacketProcessor* method), 49

clearBusy() (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 35

clearPixelConfig() (*pymepix.SPIDR.spidrdevice.SpidrDevice* method), 41

close() (*pymepix.processing.rawtodisk.Raw2Disk* method), 55

## C

calculate\_centroids\_cluster\_stream() (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* method), 55

`close_file (pymepix.processing.udpsampler.UdpSampler attribute), 56`  
`close_file (pymepix.util.tcpsampler.TcpSampler attribute), 58`  
`CloseFileCommand (pymepix.processing.datatypes.MessageType attribute), 53`  
`closeShutter() (pymepix.SPIDR.spidrcontroller.SPIDRController attribute), 30`  
`closeShutter() (pymepix.SPIDR.spidrcontroller.SPIDRController method), 35`  
`cluster_stream_preprocess() (pymepix.processing.logic.centroid_calculator.CentroidCalculator attribute), 30`  
`cluster_stream_preprocess() (pymepix.processing.logic.centroid_calculator.CentroidCalculator method), 46`  
`ClusterStream (class in pymepix.clustering.cluster_stream), 43`  
`CMD_AUTOTRIG_START (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 29`  
`CMD_AUTOTRIG_STOP (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 29`  
`CMD_BIAS_SUPPLY_ENA (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 29`  
`CMD_BURN_EFUSE (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 29`  
`CMD_CLEAR_BUSY (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 29`  
`CMD_CONFIG_CTPR (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 29`  
`CMD_DDRIVEN_READOUT (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 29`  
`CMD_DECODERS_ENA (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_DISPLAY_INFO (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_ERASE_ADDRPORTS (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_ERASE_DACS (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_ERASE_PIXCONF (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_ERASE_REGISTERS (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_ADC (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_AVDD (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_AVDD_NOW (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_BOARDID (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_CHIPBOARDID (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_CTPR (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_DAC (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_DEVICECOUNT (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_DEVICEID (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_DEVICEIDS (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_DEVICEPORT (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_DVDD (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_DVDD_NOW (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_EFUSES (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_EXTSHUTTERCNTR (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_FANSPEED (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_FIRMWVERSION (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_FPGATEMP (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_GENCONFIG (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_GPIO (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_HEADERFILTER (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_HUMIDITY (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_IPADDR_DEST (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_IPADDR_SRC (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_LOCALTEMP (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`  
`CMD_GET_OUTBLOCKCONFIG (pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30`



<i>(pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30</i>	at-	CMD_GET_VDD ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_PIXCONF ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30</i>		CMD_GET_VDD_NOW ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_PLLCONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30</i>	at-	CMD_MASK ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_PRESSURE ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30</i>		CMD_NOP ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_PWRPULSECONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 30</i>	at-	CMD_NOREPLY ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_READOUTSPEED ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_PAUSE_READOUT ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_REMOTETEMP ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_PWRPULSE_ENA ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_SERVERPORT ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_READ_FLASH ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_SHUTTERCNTR ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_REINIT_DEVICE ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_SHUTTEREND ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_REINIT_DEVICES ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_SHUTTERSTART ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_REPLY ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_SLVSCONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_RESET_COUNTERS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_SOFTVERSION ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_RESET_DEVICE ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_SPIDR_ADC ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_RESET_DEVICES ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_SPIDRREG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_RESET_MODULE ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_STARTOPTS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_RESET_PIXELS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_TIMER ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_RESET_TIMER ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_TPNUMBER ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_RESET_TIMERS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_TPPERIODPHASE ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_SELECT_CHIPBOARD ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
CMD_GET_TRIGCONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>	at-	CMD_SEQ_READOUT ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
		CMD_SET_BIAS_ADJUST ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 31</i>
		CMD_SET_BOARDID ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 32</i>
	at-	CMD_SET_BUSY ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds attribute), 32</i>

CMD_SET_CHIPBOARDID ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_SET_SERVERPORT ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_CTPR ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_SET_SLVSCONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_CTPR_LEON ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_SET_SPIDRREG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_DAC ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_SET_TIMEOFDAY ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_DACS_DFLT ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_SET_TIMER ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_EXTDAC ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_SET_TPNUMBER ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_FANSPEED ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_SET_TPPERIODPHASE ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_GENCONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_SET_TRIGCONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_GPIO ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_STORE_ADDRPORTS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_GPIO_PIN ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_STORE_DACS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_HEADERFILTER ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_STORE_PIXCONF ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_IPADDR_DEST ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_STORE_REGISTERS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_IPADDR_SRC ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_STORE_STARTOPTS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32
CMD_SET_LOGLEVEL ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_T0_SYNC ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 33
CMD_SET_OUTBLOCKCONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_TPX_POWER_ENA ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 33
CMD_SET_OUTPUTMASK ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_UPLOAD_PACKET ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 33
CMD_SET_PIXCONF ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_VALID_ADDRPORTS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 33
CMD_SET_PLLCONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_VALID_DACS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 33
CMD_SET_PWRPULSECONFIG ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_VALID_PIXCONF ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 33
CMD_SET_READOUTSPEED ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	CMD_VALID_REGISTERS ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 33
CMD_SET_SENSEDAC ( <i>pymepix.SPIDR.spidrcmds.SpidrCmds</i> attribute), 32	

CMD\_WRITE\_FLASH (*pymepix.SPIDR.spidrcmds.SpidrCmds* attribute), 33  
 columnTestPulseRegister (*pymepix.SPIDR.spidrdevice.SpidrDevice* attribute), 41  
 compute\_timewalk () (in module *pymepix.util.timewalk*), 59  
 compute\_timewalk\_lookup () (in module *pymepix.util.timewalk*), 59  
 config (*pymepix.timepixdevice.TimepixDevice* attribute), 64  
 ConfigClassException, 63  
 configureStage () (*pymepix.processing.baseacquisition.AcquisitionStage* method), 51  
 connect\_device () (*pymepix.processing.usbtrainid.USBTrainID* method), 57  
 connect\_timepix () (in module *pymepix.main*), 59  
 convertHtonl () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 35  
 convertNtohl () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 35  
 correct\_global\_time () (*pymepix.processing.logic.packet\_processor.PacketProcessor* method), 49  
 CpuToTpx (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 33  
 create\_socket\_connection () (*pymepix.processing.udpsampler.UdpSampler* method), 56  
 createConnection () (*pymepix.util.tcpsampler.TcpSampler* method), 58  
 cs\_max\_dist\_tof (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* attribute), 46  
 cs\_min\_cluster\_size (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* attribute), 46  
 cs\_sensor\_size (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* attribute), 46  
 cs\_tot\_offset (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* attribute), 46  
 CycleOverflow (*pymepix.timepixdef.TimerOverflow* attribute), 63  
 data\_callback (*pymepix.pymepix\_connection.PymepixConnection* attribute), 60  
 datadrivenReadout () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 35  
 dbscan\_clustering (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* attribute), 47  
 Default (*pymepix.SPIDR.spidrdefs.SpidrReadoutSpeed* attribute), 40  
 DefaultConfig (class in *pymepix.config.defaultconfig*), 43  
 deAcquisitionStage (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 34  
 DEFrameIDCount (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 36  
 deviceId (*pymepix.SPIDR.spidrdevice.SpidrDevice* attribute), 41  
 deviceIds (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 36  
 deviceName (*pymepix.timepixdevice.TimepixDevice* attribute), 65  
 deviceIdToString () (*pymepix.timepixdevice.TimepixDevice* method), 65  
 Disable (*pymepix.timepixdef.GrayCounter* attribute), 62  
 Disable (*pymepix.timepixdef.SuperPixel* attribute), 62  
 Disable (*pymepix.timepixdef.TestPulse* attribute), 63  
 disableExternalRefClock () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 36  
 disablePeriphClk80Mhz () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 36  
 DiscriminatorDigital (*pymepix.timepixdef.TestPulseDigAnalog* attribute), 63  
 dtot (*pymepix.processing.logic.packet\_processor.PixelOrientation* attribute), 49  
 dvdd (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 36  
 dvddNow (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 36  
 E  
 dacCodes () (*pymepix.config.defaultconfig.DefaultConfig* method), 43  
 dacCodes () (*pymepix.config.sophyconfig.SophyConfig* method), 44  
 dacCodes () (*pymepix.config.timepixconfig.TimepixConfig* method), 45  
 DacRegisterCodes (class in *pymepix.timepixdef*), 61  
 data\_thread () (*pymepix.pymepix\_connection.PymepixConnection* method), 61  
 enable (*pymepix.processing.basepipeline.BasePipelineObject* attribute), 52  
 enable (*pymepix.processing.udpsampler.UdpSampler* attribute), 56  
 Enable (*pymepix.timepixdef.GrayCounter* attribute), 62  
 Enable (*pymepix.timepixdef.SuperPixel* attribute), 62  
 Enable (*pymepix.timepixdef.TestPulse* attribute), 63

enable (*pymepix.util.tcpsampler.TcpSampler* attribute), 41  
58  
enableDecoders () (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 41  
method), 36  
enableExternalRefClock () (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 41  
method), 36  
enablePeriphClk80Mhz () (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 41  
method), 37  
enablePolling () (*pymepix.pymepix\_connection.PymepixConnection* attribute), 61  
method), 61  
epsilon (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* attribute), 47  
attribute), 47  
ERR\_ADC\_HARDW (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_DAC\_HARDW (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_FLASH\_STORAGE (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_ILLEGAL\_PAR (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_MON\_HARDW (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_MONITOR (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_MSG\_LENGTH (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_NONE (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_NOT\_IMPLEMENTED (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_SEQUENCE (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_STR (*pymepix.SPIDR.error.PymePixException* attribute), 29  
attribute), 29  
ERR\_TPX3\_HARDW (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
ERR\_UNKNOWN\_CMD (*pymepix.SPIDR.error.SPIDRErrorDefs* attribute), 29  
attribute), 29  
errorMessage () (*pymepix.SPIDR.error.PymePixException* method), 29  
method), 29  
event\_window (*pymepix.processing.logic.packet\_processor.PacketProcessor* attribute), 49  
attribute), 49  
EventData (*pymepix.processing.datatypes.MessageType* attribute), 53  
attribute), 53  
EventiTot (*pymepix.timepixdef.OperationMode* attribute), 62  
attribute), 62  
External (*pymepix.timepixdef.TestPulseGenerator* attribute), 63  
attribute), 63  
ExternalFallingRising (*pymepix.SPIDR.spidrdefs.SpidrShutterMode* attribute), 41  
attribute), 41  
ExternalFallingTimer  
ExternalRisingFalling  
ExternalRisingTimer  
ExternalShutterCounter  
filename (*pymepix.config.sophyconfig.SophyConfig* attribute), 44  
attribute), 44  
find\_events\_fast () (*pymepix.processing.logic.packet\_processor.PacketProcessor* method), 49  
method), 49  
find\_events\_fast\_post () (*pymepix.processing.logic.packet\_processor.PacketProcessor* method), 49  
method), 49  
firmwareVersion (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 37  
attribute), 37  
fpgaTemperature (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 37  
attribute), 37  
frontEndAnalog (*pymepix.timepixdef.TestPulseDigAnalog* attribute), 63  
attribute), 63

## G

genConfig (*pymepix.SPIDR.spidrdevice.SpidrDevice* attribute), 41  
attribute), 41  
get\_useful\_packets () (*pymepix.processing.udpsampler.UdpSampler* method), 56  
method), 56  
get\_useful\_packets () (*pymepix.util.tcpsampler.TcpSampler* method), 58  
method), 58  
getAdc () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 37  
method), 37  
getBuffers () (*pymepix.processing.logic.packet\_processor.PacketProcessor* method), 49  
method), 49  
getDac () (*pymepix.SPIDR.spidrdevice.SpidrDevice* method), 41  
method), 41  
getDacOut () (*pymepix.SPIDR.spidrdevice.SpidrDevice* method), 41  
method), 41  
getDevice () (*pymepix.pymepix\_connection.PymepixConnection* method), 61  
method), 61  
getLogger () (*pymepix.core.log.Logger* method), 45  
method), 45  
getLogger () (*pymepix.core.log.ProcessLogger* method), 45  
method), 45  
getPixelConfig () (*pymepix.SPIDR.spidrdevice.SpidrDevice* method), 42  
method), 42

[getSpidrReg\(\)](#) (*pymepix.SPIDR.spidrcontroller.SPIDRController* *method*), [37](#)  
[getStage\(\)](#) (*pymepix.processing.baseacquisition.AcquisitionPipeline* *method*), [51](#)  
[GrayCounter](#) (*class in pymepix.timepixdef*), [62](#)  
[grayCounter](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [65](#)  
[gui\\_bar\\_fun](#) (*pymepix.post\_processing.ProgressBar* *attribute*), [59](#)

## H

[handle\(\)](#) (*pymepix.util.spidrDummyTCP.TPX3Handler* *method*), [57](#)  
[handle\\_events](#) (*pymepix.processing.logic.packet\_processor.PacketProcessor* *attribute*), [49](#)  
[handle\\_lsb\\_time\(\)](#) (*pymepix.processing.rawfilesampler.RawFileSampler* *method*), [55](#)  
[handle\\_msb\\_time\(\)](#) (*pymepix.processing.rawfilesampler.RawFileSampler* *method*), [55](#)  
[handle\\_other\(\)](#) (*pymepix.processing.rawfilesampler.RawFileSampler* *method*), [55](#)  
[hasOutput\(\)](#) (*pymepix.processing.basepipeline.BasePipelineObject* *class method*), [52](#)  
[headerFilter](#) (*pymepix.SPIDR.spidrdevice.SpidrDevice* *attribute*), [42](#)  
[HighSpeed](#) (*pymepix.SPIDR.spidrdefs.SpidrReadoutSpeed* *attribute*), [40](#)  
[humidity](#) (*pymepix.SPIDR.spidrcontroller.SPIDRController* *attribute*), [37](#)

## I

[Ibias\\_CP\\_PLL](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_DiscS1\\_OFF](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_DiscS1\\_OFF](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_DiscS1\\_ON](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_DiscS1\\_ON](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_DiscS2\\_OFF](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_DiscS2\\_OFF](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_DiscS2\\_ON](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_DiscS2\\_ON](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_Ikrum](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_Ikrum](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_PixelDAC](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_PixelDAC](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_Preamp\\_OFF](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_Preamp\\_OFF](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_Preamp\\_ON](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_Preamp\\_ON](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_TPbufferIn](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_TPbufferIn](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[Ibias\\_TPbufferOut](#) (*pymepix.timepixdef.DacRegisterCodes* *attribute*), [61](#)  
[Ibias\\_TPbufferOut](#) (*pymepix.timepixdevice.TimepixDevice* *attribute*), [64](#)  
[init\\_new\\_process\(\)](#) (*pymepix.processing.pipeline\_packet\_processor.PipelinePacketProcessor* *method*), [54](#)  
[init\\_new\\_process\(\)](#) (*pymepix.processing.rawfilesampler.RawFileSampler* *method*), [55](#)  
[init\\_new\\_process\(\)](#) (*pymepix.processing.udpsampler.UdpSampler* *method*), [56](#)  
[init\\_new\\_process\(\)](#) (*pymepix.util.tcpsampler.TcpSampler* *method*), [58](#)  
[Internal](#) (*pymepix.timepixdef.TestPulseGenerator* *attribute*), [63](#)  
[ipAddrDest](#) (*pymepix.SPIDR.spidrdevice.SpidrDevice* *attribute*), [42](#)  
[ipAddrSrc](#) (*pymepix.SPIDR.spidrdevice.SpidrDevice* *attribute*), [42](#)  
[isAcquiring](#) (*pymepix.pymepix\_connection.PymepixConnection* *attribute*), [61](#)  
[isRunning](#) (*pymepix.processing.baseacquisition.AcquisitionPipeline* *attribute*), [51](#)

## L

[Left](#) (*pymepix.processing.logic.packet\_processor.PixelOrientation* *attribute*), [49](#)  
[linkCounts](#) (*pymepix.SPIDR.spidrcontroller.SPIDRController* *attribute*), [37](#)  
[linkStatus](#) (*pymepix.SPIDR.spidrdevice.SpidrDevice* *attribute*), [42](#)



load\_config() (in module *pymepix.config.load\_config*), 43  
loadConfig() (*pymepix.timepixdevice.TimepixDevice* method), 65  
loadFile() (*pymepix.config.sophyconfig.SophyConfig* method), 44  
localTemperature (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 37  
Logger (class in *pymepix.core.log*), 45  
LowSpeed (*pymepix.SPIDR.spidrdefs.SpidrReadoutSpeed* attribute), 40

## M

main() (in module *pymepix.config.sophyconfig*), 44  
main() (in module *pymepix.main*), 59  
main() (in module *pymepix.processing.baseacquisition*), 52  
main() (in module *pymepix.processing.basepipeline*), 53  
main() (in module *pymepix.processing.rawtodisk*), 56  
main() (in module *pymepix.processing.udpsampler*), 57  
main() (in module *pymepix.processing.usbtrainid*), 57  
main() (in module *pymepix.SPIDR.spidrcontroller*), 40  
main() (in module *pymepix.timepixdevice*), 66  
main() (in module *pymepix.util.spidrDummyTCP*), 57  
main() (in module *pymepix.util.spidrDummyUDP*), 57  
main() (in module *pymepix.util.tcpsampler*), 59  
main\_process() (in module *pymepix.processing.rawtodisk*), 56  
Mask (*pymepix.timepixdef.GrayCounter* attribute), 62  
Mask (*pymepix.timepixdef.OperationMode* attribute), 62  
Mask (*pymepix.timepixdef.SuperPixel* attribute), 62  
Mask (*pymepix.timepixdef.TestPulse* attribute), 63  
Mask (*pymepix.timepixdef.TestPulseDigAnalog* attribute), 63  
Mask (*pymepix.timepixdef.TestPulseGenerator* attribute), 63  
Mask (*pymepix.timepixdef.TimeofArrivalClock* attribute), 63  
Mask (*pymepix.timepixdef.TimerOverflow* attribute), 63  
maskPixels (*pymepix.config.defaultconfig.DefaultConfig* attribute), 43  
maskPixels (*pymepix.config.sophyconfig.SophyConfig* attribute), 44  
maskPixels (*pymepix.config.timepixconfig.TimepixConfig* attribute), 45  
MessageType (class in *pymepix.processing.datatypes*), 53  
min\_samples (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* attribute), 47  
MONITOR\_ERR\_STR (*pymepix.SPIDR.error.PymePixException* attribute), 29

## N

Negative (*pymepix.timepixdef.Polarity* attribute), 62  
numBlobProcesses (*pymepix.processing.acquisition.CentroidPipeline* attribute), 50  
numDevices (*pymepix.pymepix\_connection.PymepixConnection* attribute), 61  
numProcess (*pymepix.processing.baseacquisition.AcquisitionStage* attribute), 51

## O

Open (*pymepix.SPIDR.spidrdefs.SpidrShutterMode* attribute), 41  
open\_file() (*pymepix.processing.rawtodisk.Raw2Disk* method), 55  
open\_output\_file() (in module *pymepix.util.storage*), 58  
OpenFileCommand (*pymepix.processing.datatypes.MessageType* attribute), 53  
openShutter() (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 37  
OperationMode (class in *pymepix.timepixdef*), 62  
operationMode (*pymepix.timepixdevice.TimepixDevice* attribute), 65  
orientPixels() (*pymepix.processing.logic.packet\_processor.PacketProcessor* method), 49  
outBlockConfig (*pymepix.SPIDR.spidrdevice.SpidrDevice* attribute), 42  
outfile\_name (*pymepix.processing.udpsampler.UdpSampler* attribute), 56  
outfile\_name (*pymepix.util.tcpsampler.TcpSampler* attribute), 58  
outputQueue (*pymepix.processing.baseacquisition.AcquisitionStage* attribute), 51  
outputQueues (*pymepix.processing.basepipeline.BasePipelineObject* attribute), 53

## P

PacketProcessor (class in *pymepix.processing.logic.packet\_processor*), 48  
PacketType (class in *pymepix.timepixdef*), 62  
parseDAC() (*pymepix.config.sophyconfig.SophyConfig* method), 44  
parsePixelConfig() (*pymepix.config.sophyconfig.SophyConfig* method), 44  
pauseHeartbeat() (*pymepix.timepixdevice.TimepixDevice* method), 65  
pausedReadout() (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 37  
perform() (*pymepix.clustering.cluster\_stream.ClusterStream* method), 43  
perform\_centroiding() (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* method), 47

*method*), 47  
 perform\_centroiding\_cluster\_stream()  
     (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator*  
     *method*), 47  
 perform\_centroiding\_dbscan()  
     (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator*  
     *method*), 47  
 perform\_clustering\_dbscan() (in module  
     *pymepix.processing.logic.centroid\_calculator*),  
     48  
 perform\_clustering\_dbscan()  
     (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator*  
     *method*), 47  
 PhaseShiftedGray (*pymepix.timepixdef.TimeofArrivalClock*  
     *attribute*), 63  
 PipelineCentroidCalculator (class in  
     *pymepix.processing.pipeline\_centroid\_calculator*),  
     54  
 PipelinePacketProcessor (class in  
     *pymepix.processing.pipeline\_packet\_processor*),  
     54  
 Pixel (*pymepix.timepixdef.PacketType* *attribute*), 62  
 PixelData (*pymepix.processing.datatypes.MessageType*  
     *attribute*), 53  
 pixelMask (*pymepix.timepixdevice.TimepixDevice* *at-*  
     *tribute*), 65  
 PixelOrientation (class in  
     *pymepix.processing.logic.packet\_processor*),  
     49  
 pixelPacketCounter  
     (*pymepix.SPIDR.spidrdevice.SpidrDevice*  
     *attribute*), 42  
 PixelPipeline (class in  
     *pymepix.processing.acquisition*), 50  
 pixelTest (*pymepix.timepixdevice.TimepixDevice* *at-*  
     *tribute*), 65  
 pixelThreshold (*pymepix.timepixdevice.TimepixDevice*  
     *attribute*), 65  
 PLL\_Vcntrl (*pymepix.timepixdef.DacRegisterCodes*  
     *attribute*), 61  
 pllConfig (*pymepix.SPIDR.spidrdevice.SpidrDevice*  
     *attribute*), 42  
 Polarity (class in *pymepix.timepixdef*), 62  
 polarity (*pymepix.timepixdevice.TimepixDevice* *at-*  
     *tribute*), 65  
 poll() (*pymepix.pymepix\_connection.PymepixConnection*  
     *method*), 61  
 PollBufferEmpty, 60  
 pollBufferLength (*pymepix.pymepix\_connection.PymepixConnection*  
     *attribute*), 61  
 Positive (*pymepix.timepixdef.Polarity* *attribute*), 62  
 post\_process() (in module *pymepix.main*), 59  
 post\_process() (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator*  
     *method*), 47  
 post\_process() (*pymepix.processing.logic.packet\_processor.PacketProcessor*  
     *method*), 49  
 post\_run() (*pymepix.processing.logic.processing\_step.ProcessingStep*  
     *method*), 50  
 post\_run() (*pymepix.processing.basepipeline.BasePipelineObject*  
     *method*), 53  
 post\_run() (*pymepix.processing.pipeline\_packet\_processor.PipelinePacketProcessor*  
     *method*), 54  
 post\_run() (*pymepix.processing.rawfilesampler.RawFileSampler*  
     *method*), 55  
 post\_run() (*pymepix.processing.udpsampler.UdpSampler*  
     *method*), 56  
 post\_run() (*pymepix.util.tcpsampler.TcpSampler*  
     *method*), 58  
 powerPulseConfig (*pymepix.SPIDR.spidrdevice.SpidrDevice*  
     *attribute*), 42  
 pre\_process() (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator*  
     *method*), 47  
 pre\_process() (*pymepix.processing.logic.packet\_processor.PacketProcessor*  
     *method*), 49  
 pre\_process() (*pymepix.processing.logic.processing\_step.ProcessingStep*  
     *method*), 50  
 pre\_run() (*pymepix.processing.basepipeline.BasePipelineObject*  
     *method*), 53  
 pre\_run() (*pymepix.processing.pipeline\_packet\_processor.PipelinePacketProcessor*  
     *method*), 54  
 pre\_run() (*pymepix.processing.rawfilesampler.RawFileSampler*  
     *method*), 55  
 pre\_run() (*pymepix.processing.udpsampler.UdpSampler*  
     *method*), 56  
 pre\_run() (*pymepix.util.tcpsampler.TcpSampler*  
     *method*), 58  
 pressure (*pymepix.SPIDR.spidrcontroller.SPIDRController*  
     *attribute*), 37  
 process() (*pymepix.processing.basepipeline.BasePipelineObject*  
     *method*), 53  
 process() (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator*  
     *method*), 47  
 process() (*pymepix.processing.logic.packet\_processor.PacketProcessor*  
     *method*), 49  
 process() (*pymepix.processing.logic.processing\_step.ProcessingStep*  
     *method*), 50  
 process() (*pymepix.processing.pipeline\_centroid\_calculator.PipelineCentroidCalculator*  
     *method*), 54  
 process() (*pymepix.processing.pipeline\_packet\_processor.PipelinePacketProcessor*  
     *method*), 55  
 process\_pixels() (*pymepix.processing.logic.packet\_processor.PacketProcessor*  
     *method*), 49  
 process\_trigger1()  
     (*pymepix.processing.logic.packet\_processor.PacketProcessor*  
     *method*), 49  
 process\_trigger2()  
     (*pymepix.processing.logic.packet\_processor.PacketProcessor*  
     *method*), 49

processes (*pymepix.processing.baseacquisition.AcquisitionStage* attribute), 51  
 ProcessingStep (class in *pymepix.processing.logic.processing\_step*), 49  
 ProcessLogger (class in *pymepix.core.log*), 45  
 ProgressBar (class in *pymepix.post\_processing*), 59  
 PulseCounter (*pymepix.SPIDR.spidrdefs.SpidrShutterMode* attribute), 41  
 push\_data () (*pymepix.processing.rawfilesampler.RawFileSampler* method), 55  
 pushOutput () (*pymepix.processing.basepipeline.BasePipelineObject* method), 53  
 pymepix (module), 66  
 pymepix.clustering (module), 43  
 pymepix.clustering.cluster\_stream (module), 43  
 pymepix.config (module), 45  
 pymepix.config.defaultconfig (module), 43  
 pymepix.config.load\_config (module), 43  
 pymepix.config.sophyconfig (module), 44  
 pymepix.config.timepixconfig (module), 44  
 pymepix.core (module), 45  
 pymepix.core.log (module), 45  
 pymepix.main (module), 59  
 pymepix.post\_processing (module), 59  
 pymepix.processing (module), 57  
 pymepix.processing.acquisition (module), 50  
 pymepix.processing.baseacquisition (module), 51  
 pymepix.processing.basepipeline (module), 52  
 pymepix.processing.datatypes (module), 53  
 pymepix.processing.logic (module), 50  
 pymepix.processing.logic.centroid\_calculator (module), 46  
 pymepix.processing.logic.packet\_processor (module), 48  
 pymepix.processing.logic.processing\_step (module), 49  
 pymepix.processing.logic.shared\_processing\_parameters (module), 50  
 pymepix.processing.pipeline\_centroid\_calculator (module), 54  
 pymepix.processing.pipeline\_packet\_processor (module), 54  
 pymepix.processing.rawfilesampler (module), 55  
 pymepix.processing.rawtodisk (module), 55  
 pymepix.processing.udpsampler (module), 56  
 pymepix.processing.usbtrainid (module), 57  
 pymepix.pymepix\_connection (module), 60  
 pymepix.SPIDR (module), 43  
 pymepix.SPIDR.error (module), 29  
 pymepix.SPIDR.spidrcmds (module), 29  
 pymepix.SPIDR.spidrcontroller (module), 33  
 pymepix.SPIDR.spidrdefs (module), 40  
 pymepix.SPIDR.spidrdevice (module), 41  
 pymepix.timepixdef (module), 61  
 pymepix.timepixdevice (module), 63  
 pymepix.util (module), 59  
 pymepix.util.spidrDummyTCP (module), 57  
 pymepix.util.spidrDummyUDP (module), 57  
 pymepix.util.storage (module), 58  
 pymepix.util.tcpsampler (module), 58  
 pymepix.util.timewalk (module), 59  
 PymepixConnection (class in *pymepix.pymepix\_connection*), 60  
 PymepixException, 29

## R

Raw2Disk (class in *pymepix.processing.rawtodisk*), 55  
 RawData (*pymepix.processing.datatypes.MessageType* attribute), 53  
 RawFileSampler (class in *pymepix.processing.rawfilesampler*), 55  
 readoutSpeed (*pymepix.SPIDR.spidrdevice.SpidrDevice* attribute), 42  
 record (*pymepix.processing.udpsampler.UdpSampler* attribute), 56  
 record (*pymepix.util.tcpsampler.TcpSampler* attribute), 58  
 refreshPixels () (*pymepix.timepixdevice.TimepixDevice* method), 65  
 reinitDevice () (*pymepix.SPIDR.spidrdevice.SpidrDevice* method), 42  
 reinitDevices () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 38  
 requestGetBytes () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 38  
 requestGetInt () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 38  
 requestGetIntBytes () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 38  
 requestGetInts () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 38  
 requestSetInt () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 38  
 requestSetIntBytes () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 38



`method`), 38  
`requestSetInts()` (`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 38  
`reset()` (`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`resetCounters()` (`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 38  
`resetDevices()` (`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 38  
`resetModule()` (`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 38  
`resetPacketCounters()`  
(`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 39  
`resetPixelConfig()`  
(`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`resetPixels()` (`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`resetPixels()` (`pymepix.timepixdevice.TimepixDevice`  
`method`), 65  
`resetTimers()` (`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 39  
`restartTimers()` (`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 39  
`resumeHeartbeat()`  
(`pymepix.timepixdevice.TimepixDevice`  
`method`), 65  
`Right` (`pymepix.processing.logic.packet_processor.PixelOrientation`  
`attribute`), 49  
`run()` (`pymepix.processing.basepipeline.BasePipelineObject`  
`method`), 53  
`run()` (`pymepix.processing.rawfilesampler.RawFileSampler`  
`method`), 55  
`run()` (`pymepix.processing.udpsampler.UdpSampler`  
`method`), 57  
`run()` (`pymepix.processing.usbtrainid.USBTrainID`  
`method`), 57  
`run()` (`pymepix.util.tcpsampler.TcpSampler` `method`),  
59  
`run_post_processing()` (`in module`  
`pymepix.post_processing`), 59  
**S**  
`saveMask()` (`pymepix.config.sophyconfig.SophyConfig`  
`method`), 44  
`saveToHDF5()` (`pymepix.processing.rawfilesampler.RawFileSampler`  
`method`), 55  
`sequentialReadout()`  
(`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 39  
`serverPort` (`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`attribute`), 42  
`setArgs()` (`pymepix.processing.baseacquisition.AcquisitionStage`  
`method`), 51  
`setBiasSupplyEnable()`  
(`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 39  
`setConfigClass()` (`pymepix.timepixdevice.TimepixDevice`  
`method`), 65  
`setDac()` (`pymepix.timepixdevice.TimepixDevice`  
`method`), 65  
`setDacDefault()` (`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setEthernetFilter()`  
(`pymepix.timepixdevice.TimepixDevice`  
`method`), 65  
`setExternalDac()` (`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setHeaderFilter()`  
(`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setPixelMask()` (`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setPixelMask()` (`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setPixelTestBit()`  
(`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setPixelThreshold()`  
(`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setPowerPulseEnable()`  
(`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 39  
`setSenseDac()` (`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setShutterTriggerConfig()`  
(`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 39  
`setSinglePixelMask()`  
(`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setSinglePixelTestBit()`  
(`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setSinglePixelThreshold()`  
(`pymepix.SPIDR.spidrdevice.SpidrDevice`  
`method`), 42  
`setSpidrReg()` (`pymepix.SPIDR.spidrcontroller.SPIDRController`  
`method`), 39  
`setTpPeriodPhase()`  
(`pymepix.SPIDR.spidrdevice.SpidrDevice`

<code>method</code> ), 42	( <code>pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>setTpxPowerPulseEnable()</code>	<code>tribute</code> ), 40	
<code>(pymepix.SPIDR.spidrcontroller.SPIDRControllerSPIDR_IPMUX_CONFIG_I</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>method</code> ), 39	<code>tribute</code> ), 40	
<code>setupAcquisition()</code>	<code>(pymepix.timepixdevice.TimepixDevice</code>	
<code>method</code> ), 66	<code>SPIDR_PIXEL_FILTER_I</code>	
<code>setupDevice()</code> ( <code>pymepix.timepixdevice.TimepixDevice</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>method</code> ), 66	<code>tribute</code> ), 40	
<code>SharedProcessingParameter</code> (class in	<code>SPIDR_PIXEL_PKT_COUNTER_I</code>	
<code>pymepix.processing.logic.shared_processing_parameter</code> ),	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
50	<code>tribute</code> ), 40	
<code>shutterCounter</code> ( <code>pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>SPIDR_PIXEL_PKT_COUNTER_OLD_I</code>	
<code>attribute</code> ), 39	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>shutterEnd</code> ( <code>pymepix.SPIDR.spidrdevice.SpidrDevice</code>	<code>tribute</code> ), 40	
<code>attribute</code> ), 42	<code>SPIDR_SHUTTERTRIG_CNT_I</code>	
<code>shutterStart</code> ( <code>pymepix.SPIDR.spidrdevice.SpidrDevice</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>attribute</code> ), 42	<code>tribute</code> ), 40	
<code>shutterTriggerConfig</code>	<code>SPIDR_SHUTTERTRIG_CTRL_I</code>	
<code>(pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>attribute</code> ), 39	<code>tribute</code> ), 40	
<code>ShutterTriggerCount</code>	<code>SPIDR_SHUTTERTRIG_DELAY_I</code>	
<code>(pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>attribute</code> ), 34	<code>tribute</code> ), 40	
<code>ShutterTriggerCtrl</code>	<code>SPIDR_SHUTTERTRIG_FREQ_I</code>	
<code>(pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>attribute</code> ), 34	<code>tribute</code> ), 40	
<code>ShutterTriggerDelay</code>	<code>SPIDR_SHUTTERTRIG_LENGTH_I</code>	
<code>(pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>attribute</code> ), 34	<code>tribute</code> ), 41	
<code>ShutterTriggerFreq</code>	<code>SPIDR_TDC_TRIGGER_COUNTER_I</code>	
<code>(pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>attribute</code> ), 34	<code>tribute</code> ), 41	
<code>ShutterTriggerLength</code>	<code>SPIDR_UDP_PKT_COUNTER_I</code>	
<code>(pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>attribute</code> ), 34	<code>tribute</code> ), 41	
<code>ShutterTriggerMode</code>	<code>SPIDR_UDPMON_PKT_COUNTER_I</code>	
<code>(pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>attribute</code> ), 35	<code>tribute</code> ), 41	
<code>slaveConfig</code> ( <code>pymepix.SPIDR.spidrdevice.SpidrDevice</code>	<code>SPIDR_UDPPAUSE_PKT_COUNTER_I</code>	
<code>attribute</code> ), 42	<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	at-
<code>softwareVersion</code> ( <code>pymepix.SPIDR.spidrcontroller.SPIDRController</code>	<code>tribute</code> ), 41	
<code>attribute</code> ), 39	<code>SPIDR_UDP_COUNTER_I</code>	
<code>SophyConfig</code> (class in <code>pymepix.config.sophyconfig</code> ),	<code>SPIDRController</code> (class in <code>pymepix.SPIDR.spidrcmds</code> ), 29	
44	<code>SPIDRController</code> (class in	
	<code>pymepix.SPIDR.spidrcontroller</code> ), 33	
	<code>SpidrDevice</code> (class in <code>pymepix.SPIDR.spidrdevice</code> ),	
	41	
<code>SPIDR_CPU2TPX_WR_I</code>	<code>SPIDR_ErrorDefs</code> (class in <code>pymepix.SPIDR.error</code> ), 29	
<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	<code>spidrFanSpeed</code> ( <code>pymepix.SPIDR.spidrcontroller.SPIDRController</code>	
<code>tribute</code> ), 40	<code>attribute</code> ), 40	
<code>SPIDR_DEVICES_AND_PORTS_I</code>	<code>SpidrReadoutSpeed</code> (class in	
<code>(pymepix.SPIDR.spidrdefs.SpidrRegs</code>	<code>pymepix.SPIDR.spidrdefs</code> ), 40	in
<code>tribute</code> ), 40	<code>SpidrRegs</code> (class in <code>pymepix.SPIDR.spidrdefs</code> ), 40	
<code>SPIDR_ERR_STR</code> ( <code>pymepix.SPIDR.error.PymePixException</code>	<code>SpidrShutterMode</code> (class in	
<code>attribute</code> ), 29	<code>pymepix.SPIDR.spidrdefs</code> ), 41	
<code>SPIDR_FE_GTX_CTRL_STAT_I</code>		

stage (*pymepix.processing.baseacquisition.AcquisitionStage* attribute), 42  
 stage (*pymepix.processing.baseacquisition.AcquisitionStage* attribute), 51  
 stages (*pymepix.processing.baseacquisition.AcquisitionPipeline* attribute), 51  
 start () (*pymepix.processing.baseacquisition.AcquisitionPipeline* attribute), 35  
 start () (*pymepix.processing.baseacquisition.AcquisitionPipeline* method), 51  
 start () (*pymepix.processing.baseacquisition.AcquisitionStage* attribute), 43  
 start () (*pymepix.processing.baseacquisition.AcquisitionStage* method), 52  
 start () (*pymepix.pymepix\_connection.PymepixConnection* attribute), 44  
 start () (*pymepix.pymepix\_connection.PymepixConnection* method), 61  
 start () (*pymepix.timepixdevice.TimepixDevice* attribute), 45  
 start () (*pymepix.timepixdevice.TimepixDevice* method), 66  
 start\_recording () (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 start\_recording () (*pymepix.timepixdevice.TimepixDevice* method), 66  
 startAutoTrigger () (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 40  
 startAutoTrigger () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 40  
 startTrainID () (*pymepix.processing.baseacquisition.AcquisitionPipeline* attribute), 52  
 startTrainID () (*pymepix.processing.baseacquisition.AcquisitionPipeline* method), 52  
 stop () (*pymepix.processing.baseacquisition.AcquisitionPipeline* attribute), 51  
 stop () (*pymepix.processing.baseacquisition.AcquisitionPipeline* method), 51  
 stop () (*pymepix.processing.baseacquisition.AcquisitionStage* attribute), 52  
 stop () (*pymepix.processing.baseacquisition.AcquisitionStage* method), 52  
 stop () (*pymepix.pymepix\_connection.PymepixConnection* attribute), 61  
 stop () (*pymepix.pymepix\_connection.PymepixConnection* method), 61  
 stop () (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 stop () (*pymepix.timepixdevice.TimepixDevice* method), 66  
 stop\_recording () (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 stop\_recording () (*pymepix.timepixdevice.TimepixDevice* method), 66  
 stopAutoTrigger () (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 40  
 stopAutoTrigger () (*pymepix.SPIDR.spidrcontroller.SPIDRController* method), 40  
 StopOverflow (*pymepix.timepixdef.TimerOverflow* attribute), 63  
 stopRaw2Disk () (*pymepix.util.tcpsampler.TcpSampler* attribute), 59  
 stopRaw2Disk () (*pymepix.util.tcpsampler.TcpSampler* method), 59  
 stopTrainID () (*pymepix.processing.baseacquisition.AcquisitionStage* attribute), 52  
 stopTrainID () (*pymepix.processing.baseacquisition.AcquisitionStage* method), 52  
 store\_centroid () (in module *pymepix.util.storage*), 58  
 STORE\_ERR\_STR (*pymepix.SPIDR.error.PymePixException* attribute), 29  
 store\_raw () (in module *pymepix.util.storage*), 58  
 store\_toa () (in module *pymepix.util.storage*), 58  
 store\_tof () (in module *pymepix.util.storage*), 58  
 SuperPixel (class in *pymepix.timepixdef*), 62  
 superPixel (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 SystemClock (*pymepix.timepixdef.TimeofArrivalClock* attribute), 63  
 SystemClock (*pymepix.timepixdef.TimeofArrivalClock* method), 63  
 TcpSampler (class in *pymepix.util.tcpsampler*), 58  
 TcpSamplerCounter (*pymepix.SPIDR.spidrcontroller.SPIDRController* attribute), 40  
 testPixels (*pymepix.config.defaultconfig.DefaultConfig* attribute), 43  
 testPixels (*pymepix.config.sophyconfig.SophyConfig* attribute), 44  
 testPixels (*pymepix.config.timepixconfig.TimepixConfig* attribute), 45  
 TestPulse (class in *pymepix.timepixdef*), 62  
 testPulse (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 TestPulseDigAnalog (class in *pymepix.timepixdef*), 63  
 testPulseDigitalAnalog (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 TestPulseGenerator (class in *pymepix.timepixdef*), 63  
 testPulseGeneratorSource (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 thresholdPixels (*pymepix.config.defaultconfig.DefaultConfig* attribute), 43  
 thresholdPixels (*pymepix.config.sophyconfig.SophyConfig* attribute), 44  
 thresholdPixels (*pymepix.config.timepixconfig.TimepixConfig* attribute), 45  
 TimeofArrivalClock (class in *pymepix.timepixdef*), 63  
 timeOfArrivalClock (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 TimepixConfig (class in *pymepix.config.timepixconfig*), 44  
 TimepixDevice (class in *pymepix.timepixdevice*), 63  
 timer (*pymepix.SPIDR.spidrdevice.SpidrDevice* attribute), 42  
 TimerOverflow (class in *pymepix.timepixdef*), 63  
 TimerOverflowControl (*pymepix.timepixdevice.TimepixDevice* attribute), 66  
 ToA (*pymepix.timepixdef.OperationMode* attribute), 62  
 ToAandToT (*pymepix.timepixdef.OperationMode* attribute), 62  
 tot\_threshold (*pymepix.processing.logic.centroid\_calculator.CentroidCalculator* attribute), 47  
 tpNumber (*pymepix.SPIDR.spidrdevice.SpidrDevice* attribute), 42  
 TpPeriodPhase (*pymepix.SPIDR.spidrdevice.SpidrDevice* attribute), 41  
 TPX3\_ERR\_STR (*pymepix.SPIDR.error.PymePixException* attribute), 29

## T

attribute), 29  
 TPX3Handler (class in `pymepix.util.spidrDummyTCP`), 57  
 Trigger (`pymepix.timepixdef.PacketType` attribute), 62  
 TriggerData (`pymepix.processing.datatypes.MessageType` attribute), 54  
 triggers\_processed (`pymepix.processing.logic.centroid_calculator.CentroidCalculator` attribute), 47  
**U**  
 UdpMonPacketCounter (`pymepix.SPIDR.spidrcontroller.SPIDRController` attribute), 35  
 UdpPacketCounter (`pymepix.SPIDR.spidrcontroller.SPIDRController` attribute), 35  
 UdpPausePacketCounter (`pymepix.SPIDR.spidrcontroller.SPIDRController` attribute), 35  
 UdpSampler (class in `pymepix.processing.udpsampler`), 56  
 UnknownParameterTypeException, 50  
 Up (`pymepix.processing.logic.packet_processor.PixelOrientation` attribute), 49  
 update\_timer() (`pymepix.timepixdevice.TimepixDevice` method), 66  
 update\_to() (`pymepix.post_processing.ProgressBar` method), 59  
 updateBuffers() (`pymepix.processing.logic.packet_processor.PacketProcessor` method), 49  
 updateProgressBar() (in module `pymepix.post_processing`), 60  
 uploadPacket() (`pymepix.SPIDR.spidrdevice.SpidrDevice` method), 42  
 uploadPixelConfig() (`pymepix.SPIDR.spidrdevice.SpidrDevice` method), 42  
 uploadPixels() (`pymepix.timepixdevice.TimepixDevice` method), 66  
 USBTrainID (class in `pymepix.processing.usbtrainid`), 57  
**V**  
 value (`pymepix.processing.logic.shared_processing_parameter.SharedProcessingParameter` attribute), 50  
 vdd (`pymepix.SPIDR.spidrcontroller.SPIDRController` attribute), 40  
 vddNow (`pymepix.SPIDR.spidrcontroller.SPIDRController` attribute), 40  
 Vfbk (`pymepix.timepixdef.DacRegisterCodes` attribute), 62  
 Vfbk (`pymepix.timepixdevice.TimepixDevice` attribute), 64  
 VPreamp\_NCAS (`pymepix.timepixdef.DacRegisterCodes` attribute), 62  
 VPreamp\_NCAS (`pymepix.timepixdevice.TimepixDevice` attribute), 64  
 Vthreshold\_coarse (`pymepix.timepixdef.DacRegisterCodes` attribute), 62  
 Vthreshold\_coarse (`pymepix.timepixdevice.TimepixDevice` attribute), 64  
 Vthreshold\_fine (`pymepix.timepixdef.DacRegisterCodes` attribute), 62  
 Vthreshold\_fine (`pymepix.timepixdevice.TimepixDevice` attribute), 64  
 VTP\_coarse (`pymepix.timepixdevice.TimepixDevice` attribute), 64  
 VTP\_fine (`pymepix.timepixdef.DacRegisterCodes` attribute), 62  
 VTP\_fine (`pymepix.timepixdevice.TimepixDevice` attribute), 64  
**W**  
 write() (`pymepix.processing.rawtodisk.Raw2Disk` method), 56