# idelib Documentation

***Release 3.2.3***

**David Randall Stokes**

**Jan 18, 2022**

# CONTENTS:

*idelib* is a Python API for accessing enDAQ's IDE recordings. The IDE format is an EBML encoded file using a custom schema. This library utilizes our ebmlite to parse the files, and provides classes that make reading data simple.

CHAPTER

ONE

# IDE FILE BASICS

## 1.1 What's an IDE file?

An IDE file is a read-only hierarchical data format that stores recording information generated by an enDAQ sensor device. It contains both time-indexed data from several different kinds of sensors (like acceleration, pressure, temperature, etc.), as well as metadata about the recording device (like device serial number, model number, device name, etc.) and recording settings.

## 1.2 Accessing an IDE file

The top-level interface for an IDE file is the `Dataset` object, through which one can access all of the above-listed information. When you open a file for reading, for example, this is the type of object that is returned.

### 1.2.1 Opening an IDE File

You can open an IDE file like so:

```
filename = "/path/to/your/file.IDE"
with idelib.importFile(filename) as ds:
    print(type(ds))
```

Note: a `Dataset` object perfoms *lazy-loading*, meaning that it only loads information as is needed. As a result, it internally retains a handle to the source file which needs to be closed after use. This can be accomplished by either:

- using `Dataset` as a *context manager* (as seen above; this is the recommended method), or
- by using `Dataset` as a normal object and calling the `close()` method manually:

  ```
  filename = "/path/to/your/file.IDE"
  ds = idelib.importFile(filename)
  # use `ds` here
  ds.close()  # remember to close the file after use!
  ```

## 1.3 Getting recording data

### 1.3.1 Channels & subchannels

IDE files organize recording data into *channels* and *subchannels*. A channel encapsulates data recorded by a particular individual sensor on the device (e.g., XYZ acceleration from the ADXL375 DC Accelerometer); a subchannel, if present, specifies a particular data stream within a channel (e.g., the X-coordinate acceleration from the ADXL375 DC Accelerometer).

At the top-level, a `Dataset` object has a `channels` member, which is a dict of all channels recorded in the file. The dict is keyed by channel ID numbers, with `Channel` objects in the values.

Each `Channel` object has a `subchannels` member, which is a list of `Subchannel` objects. If the channel has no subchannels, this member will be `None`.

The channels, channel ID numbers and subchannels that may appear in a given recording file depend on the physical sensors available on the recording device, which are indicated by the device's *product number*. Below are some product number abbreviations used herein:

| (Abbreviated) Product No. | Description | Example Product Nos. |
|---|---|---|
| S/W-D | enDAQ S-series & W-series devices with a single digital accelerometer | S3-D16, W5-D40 |
| S/W-DD | enDAQ S-series & W-series devices with dual digital accelerometers | S1-D100D40, S2-D25D16 |
| S/W-ED | enDAQ S-series & W-series devices with an analog electrocapacitive and a digital accelerometer | W8-E25D40, S4-E100D40 |
| S/W-RD | enDAQ S-series & W-series devices with an analog piezoresistive and a digital accelerometer | S4-R500D40, W8-R2000D40 |
| SSX | Mide Slam Stick X data recorders | SSX |
| SSC | Mide Slam Stick C data recorders | SSC |
| SSS | Mide Slam Stick S data recorders | SSS |

The below table lists channel ID numbers used in a recording file based on the recording device's product number (device series numbers and accelerometer sensitivity ranges are omitted when applicable to all such devices):

| Sensor | Channel ID | Valid Devices | Subchannels |
|---|---|---|---|
| Main Accelerometer | 8 | S/W-RD, S/W-ED, SSS, SSX | X-, Y-, Z-axis Acceleration |
| 16/200g Accelerometer | 32 | S/W-DD, SSX, SSS, SSC, S/W-D16, S/W-D200 | X-, Y-, Z-axis Acceleration |
| 8/40g Accelerometer | 80 | S/W-RD, S/W-DD, S/W-ED, S/W-D40, S/W-D8 | X-, Y-, Z-axis Acceleration |
| IMU Gyroscope | 47 | All[1] | X-, Y-, Z-axis Rotation |
| Absolute Orientation | 65 | All[Page 5, 1] | X-, Y-, Z-, W-axis Quaternion; Acc |
| Relative Orientation | 70 | All[Page 5, 1] | X-, Y-, Z-, W-axis Quaternion |
| MPL3115 | 36 | All[Page 5, 1] | Pressure, Temperature[2] |
| MS8607 | 59 | All[?] | Pressure, Temperature, Humidity[3] |
| SI1133 | 76 | All[?] | Lux, UV |

To simply use all recording data, we can iterate through each subchannel in a dataset like so:

```
for ch in ds.channels.values():
    for sch in ch.subchannels:
        print(sch)
```

### 1.3.2 EventArrays & raw data

Each `Channel` and `SubChannel` object has a `getSession()` method, which returns an `EventArray` object. `EventArray` is a wrapper around a channel's underlying recording data that loads data on demand from the source file. You can index an `EventArray` (e.g., `eventarray[i]` for some index `i`) to get a numpy `ndarray` of data. Data is organized in an n-dimensional array.

For subchannels, this will always be a 2-by-n array, where n is the number of samples recorded; `eventarray[1]` indexes the samples, `eventarray[0]` indexes the respective timestamps.

For channels, this will be a (c+1)-by-n array, where n is the number of samples recorded and c is the number of subchannels; `eventarray[1:]` indexes the samples, `eventarray[0]` indexes the respective timestamps.

## 1.4 Getting metadata

`Dataset` makes available some basic metadata. Some useful pieces of information are stored directly as members:

```
>>> ds.filename
'/home/enDAQ/recordings/test.IDE'
```

Other data is stored in the dict member `recorderInfo`:

```
>>> ds.recorderInfo['RecorderSerial']
10118
>>> ds.recorderInfo['PartNumber']
'W8-E100D40'
```

`EventArray` also stores some sample-specific metadata, like the data's units:

```
>>> eventarray.units
('Acceleration', 'g')
```

[1] excluding early SSC/SSS/SSX models
[2] 1 Hz Internal Measurements
[3] 10 Hz Control Pad Measurements

CHAPTER

TWO

# *IDELIB* API REFERENCE

## 2.1 dataset.py

Module for reading and analyzing Mide Instrumentation Data Exchange (MIDE) files.

Created on Sep 26, 2013

**author** dstokes

### 2.1.1 Classes

**class** idelib.dataset.**Dataset**(*stream*, *name=None*, *quiet=True*, *attributes=None*)

A collection of sensor data and associated configuration info. Typically represents a single MIDE EMBL file.

Dictionary attributes are all keyed by the relevant ID (sensor ID, channel ID, etc.).

**Variables**

- **loading** – Boolean; *True* if a file is still loading (or has not yet been loaded).
- **fileDamaged** – Boolean; *True* if the file ended prematurely.
- **loadCancelled** – Boolean; *True* if the file loading was aborted part way through.
- **sessions** – A list of individual Session objects in the data set. A valid file will have at least one, even if there are no *Session* elements in the data.
- **sensors** – A dictionary of Sensors.
- **channels** – A dictionary of individual Sensor channels.
- **plots** – A dictionary of individual Plots, the modified output of a Channel (or even another plot).
- **transforms** – A dictionary of functions (or function-like objects) for adjusting/calibrating sensor data.

Constructor. Typically, these objects will be instantiated by functions in the *importer* module.

**Parameters**

- **stream** – A file-like stream object containing EBML data.
- **name** – An optional name for the Dataset. Defaults to the base name of the file (if applicable).
- **quiet** – If *True*, non-fatal errors (e.g. schema/file version mismatches) are suppressed.
- **attributes** – A dictionary of arbitrary attributes, e.g. `Attribute` elements parsed from the file. Typically used for diagnostic data.

addChannel(*channelId=None*, *parser=None*, *channelClass=None*, ***kwargs*)
Add a Channel to a Sensor. Note that the *channelId* and *parser* keyword arguments are *not* optional.

**Parameters**

- **channelId** – An unique ID number for the channel.
- **parser** – The Channel's data parser
- **channelClass** – An alternate (sub)class of channel. Defaults to *None*, which creates a standard *Channel*.

addSensor(*sensorId=None*, *name=None*, *sensorClass=None*, *traceData=None*, *transform=None*, *attributes=None*, *bandwidthLimitId=None*)
Create a new Sensor object, and add it to the dataset, and return it. If the given sensor ID already exists, the existing sensor is returned instead. To modify a sensor or add a sensor object created elsewhere, use *Dataset.sensors[sensorId]* directly.

Note that the *sensorId* keyword argument is *not* optional.

**Parameters**

- **sensorId** – The ID of the new sensor.
- **name** – The new sensor's name
- **sensorClass** – An alternate (sub)class of sensor. Defaults to *None*, which creates a *Sensor*.

**Returns** The new sensor.

addSession(*startTime=None*, *endTime=None*, *utcStartTime=None*)
Create a new session, add it to the Dataset, and return it. Part of the import process.

addTransform(*transform*)
Add a transform (calibration, etc.) to the dataset. Various child objects will reference them by ID. Note: unlike the other *add* methods, this does not instantiate new objects.

addWarning(*warningId=None*, *channelId=None*, *subchannelId=None*, *low=None*, *high=None*, ***kwargs*)
Add a *WarningRange* to the dataset, which indicates when a sensor is reporting values outside of a given range.

**Parameters**

- **warningId** – A unique numeric ID for the *WarningRange*.
- **channelId** – The channel ID of the source being monitored.
- **subchannelId** – The monitored source's subchannel ID.
- **low** – The minimum value of the acceptable range.
- **high** – The maximum value of the acceptable range.

**Returns** The new *WarningRange* instance.

property channels
A dictionary of individual Sensor channels.

close()
Close the recording file.

property closed
Has the recording file been closed?

**endSession**()
Set the current session's start/end times. Part of the import process.

**property exitCondition**
The numeric code number for the condition that stopped the recording.

**getPlots**(*subchannels=True*, *plots=True*, *debug=True*, *sort=True*)
Get all plotable data sources: sensor SubChannels and/or Plots.

**Parameters**

- **subchannels** – Include subchannels if *True*.
- **plots** – Include Plots if *True*.
- **debug** – If *False*, exclude debugging/diagnostic channels.
- **sort** – Sort the plots by name if *True*.

**hasSession**(*sessionId*)
Does the Dataset contain a specific session number?

**hierarchy**()
Get a list of parents/grandparents all the way back to the root. The root is the first item in the list.

**property lastSession**
Retrieve the latest Session.

**path**()
Get the combined names of all the object's parents/grandparents.

**updateTransforms**()
Update the transforms (e.g. the calibration functions) in this dataset. This should be called before utilizing data in the set.

class idelib.dataset.**Channel**(*dataset*, *channelId=None*, *parser=None*, *sensor=None*, *name=None*, *units=None*, *transform=None*, *displayRange=None*, *sampleRate=None*, *cache=False*, *singleSample=None*, *attributes=None*)
Output from a Sensor, containing one or more SubChannels. A Sensor contains one or more Channels. SubChannels of a Channel can be accessed by index like a list or tuple.

**Variables**

- **types** – A tuple with the type of data in each of the Channel's Subchannels.
- **displayRange** – The possible ranges of each subchannel, dictated by the parser. Not necessarily the same as the range of actual values recorded in the file!

Constructor. This should generally be done indirectly via *Dataset.addChannel()*.

**Parameters**

- **sensor** – The parent sensor, if this Channel contains only data from a single sensor.
- **channelId** – The channel's ID, unique within the file.
- **parser** – The channel's EBML data parser.
- **name** – A custom name for this channel.
- **units** – The units measured in this channel, used if units are not explicitly indicated in the Channel's SubChannels.
- **transform** – A Transform object for adjusting sensor readings at the Channel level.
- **displayRange** – A 'hint' to the minimum and maximum values of data in this channel.

- **cache** – If *True*, this channel's data will be kept in memory rather than lazy-loaded.
- **singleSample** – A 'hint' that the data blocks for this channel each contain only a single sample (e.g. temperature/ pressure on an SSX). If *None*, this will be determined from the sample data.
- **attributes** – A dictionary of arbitrary attributes, e.g. `Attribute` elements parsed from the file.

**addSubChannel**(*subchannelId=None*, *channelClass=None*, ***kwargs*)
Create a new SubChannel of the Channel.

**getSession**(*sessionId=None*)
Retrieve data recorded in a Session.

**Parameters** **sessionId** – The ID of the session to retrieve.

**Returns** The recorded data.

**Return type** *EventArray*

**getSubChannel**(*subchannelId*)
Retrieve one of the Channel's SubChannels. All Channels have at least one. A SubChannel object will be automatically generated if one hasn't already explicitly been defined.

**Parameters** **subchannelId** –

**Returns** The SubChannel matching the given ID.

**getTransforms**(*id_=None*, *_tlist=None*)
Get a list of all transforms applied to the data, from first (the lowest-level parent) to last (the transform, if any, on the object itself).

**hierarchy**()
Get a list of parents/grandparents all the way back to the root. The root is the first item in the list.

**parseBlock**(*block*, *start=None*, *end=None*, *step=1*, *subchannel=None*)
Parse subsamples out of a data block. Used internally.

**Parameters**

- **block** – The data block from which to parse subsamples.
- **start** – The first block index to retrieve.
- **end** – The last block index to retrieve.
- **step** – The number of steps between samples.
- **subchannel** – If supplied, return only the values for a specific subchannel (i.e. the method is being called by a SubChannel).

**Returns** A list of tuples, one for each subsample.

**parseBlockByIndex**(*block*, *indices*, *subchannel=None*)

**Convert raw data into a set of subchannel values, returning only** specific items from the result by index.

**Parameters**

- **block** – The data block element to parse.
- **indices** – A list of sample index numbers to retrieve.

- **subchannel** – If supplied, return only the values for a specific subchannel

**Returns** A list of tuples, one for each subsample.

**path**()
Get the combined names of all the object's parents/grandparents.

**setTransform**(*transform*, *update=True*)
Set the transforming function/object. This does not change the value of *raw*, however; the new transform will not be applied unless it is *True*.

**updateTransforms**()
Recompute cached transform functions.

class idelib.dataset.**SubChannel**(*parent*, *subchannelId*, *name=None*, *units=(", ")*, *transform=None*, *displayRange=None*, *sensorId=None*, *warningId=None*, *axisName=None*, *attributes=None*, *color=None*)

Output from a sensor, derived from a channel containing multiple pieces of data (e.g. the Y from an accelerometer's XYZ). Looks like a 'real' channel.

Constructor. This should generally be done indirectly via *Channel.addSubChannel()*.

**Parameters**

- **sensor** – The parent sensor.
- **channelId** – The channel's ID, unique within the file.
- **parser** – The channel's payload data parser.
- **name** – A custom name for this channel.
- **units** – The units measured in this channel, used if units are not explicitly indicated in the Channel's SubChannels. A tuple containing the 'axis name' (e.g. 'Acceleration') and the unit symbol ('g').
- **transform** – A Transform object for adjusting sensor readings at the Channel level.
- **displayRange** – A 'hint' to the minimum and maximum values of data in this channel.
- **sensorId** – The ID of the sensor that generates this SubChannel's data.
- **warningId** – The ID of the *WarningRange* that indicates conditions that may adversely affect data recorded in this SubChannel.
- **axisName** – The name of the axis this SubChannel represents. Use if the *name* contains additional text (e.g. "X" if the name is "Accelerometer X (low-g)").
- **attributes** – A dictionary of arbitrary attributes, e.g. `Attribute` elements parsed from the file.

**addSubChannel**(**args*, ***kwargs*)
Create a new SubChannel of the Channel.

**getSession**(*sessionId=None*)
Retrieve a session by ID. If none is provided, the last session in the Dataset is returned.

**getSubChannel**(**args*, ***kwargs*)
Retrieve one of the Channel's SubChannels. All Channels have at least one. A SubChannel object will be automatically generated if one hasn't already explicitly been defined.

**Parameters** **subchannelId** –

**Returns** The SubChannel matching the given ID.

**getTransforms**(*id_=None*, *_tlist=None*)
Get a list of all transforms applied to the data, from first (the lowest-level parent) to last (the transform, if any, on the object itself).

**hierarchy**()
Get a list of parents/grandparents all the way back to the root. The root is the first item in the list.

**parseBlock**(*block*, *start=None*, *end=None*, *step=1*)
Parse subsamples out of a data block. Used internally.

**Parameters**

- **block** – The data block from which to parse subsamples.
- **start** – The first block index to retrieve.
- **end** – The last block index to retrieve.
- **step** – The number of steps between samples.

**parseBlockByIndex**(*block*, *indices*)
Parse specific subsamples out of a data block. Used internally.

**Parameters**

- **block** – The data block from which to parse subsamples.
- **indices** – A list of individual index numbers to get.

**path**()
Get the combined names of all the object's parents/grandparents.

**setTransform**(*transform*, *update=True*)
Set the transforming function/object. This does not change the value of *raw*, however; the new transform will not be applied unless it is *True*.

**updateTransforms**()
Recompute cached transform functions.

class idelib.dataset.**EventArray**(*parentChannel*, *session=None*, *parentList=None*)
A list-like object containing discrete time/value pairs. Data is dynamically read from the underlying EBML file.

Constructor. This should almost always be done indirectly via the *getSession()* method of *Channel* and *SubChannel* objects.

**append**(*block*)
Add one data block's contents to the Channel's list of data. Note that this doesn't double-check the channel ID specified in the data, but it is inadvisable to include data from different channels.

**Attention** Added elements must be in chronological order!

**arrayJitterySlice**(*start=None*, *end=None*, *step=1*, *jitter=0.5*, *display=False*)
Create an array of events within a range of indices.

**Parameters**

- **start** – The first index in the range, or a slice.
- **end** – The last index in the range. Not used if *start* is a slice.
- **step** – The step increment. Not used if *start* is a slice.
- **jitter** – The amount by which to vary the sample time, as a normalized percentage of the regular time between samples.

- **display** – If *True*, the *EventArray* transform (i.e. the 'display' transform) will be applied to the data.

**Returns** a structured array of events in the specified index range.

**arrayMinMeanMax**(*startTime=None*, *endTime=None*, *padding=0*, *times=True*, *display=False*, *iterator=<built-in function iter>*)

Get the minimum, mean, and maximum values for blocks within a specified interval.

**Todo** Remember what *padding* was for, and either implement or remove it completely. Related to plotting; see *plots*.

**Parameters**

- **startTime** – The first time (in microseconds by default), *None* to start at the beginning of the session.
- **endTime** – The second time, or *None* to use the end of the session.
- **times** – If *True* (default), the results include the block's starting time.
- **display** – If *True*, the final 'display' transform (e.g. unit conversion) will be applied to the results.

**Returns** A structured array of data block statistics (min, mean, and max, respectively).

**arrayRange**(*startTime=None*, *endTime=None*, *step=1*, *display=False*)

Get a set of data occurring in a given time interval.

**Parameters**

- **startTime** – The first time (in microseconds by default), *None* to start at the beginning of the session.
- **endTime** – The second time, or *None* to use the end of the session.

**Returns** a structured array of events in the specified time interval.

**arrayResampledRange**(*startTime*, *stopTime*, *maxPoints*, *padding=0*, *jitter=0*, *display=False*)

Retrieve the events occurring within a given interval, undersampled as to not exceed a given length (e.g. the size of the data viewer's screen width).

**Todo** Optimize iterResampledRange(); not very efficient, particularly not with single-sample blocks.

**arraySlice**(*start=None*, *end=None*, *step=1*, *display=False*)

Create an array of events within a range of indices.

**Parameters**

- **start** – The first index in the range, or a slice.
- **end** – The last index in the range. Not used if *start* is a slice.
- **step** – The step increment. Not used if *start* is a slice.
- **display** – If *True*, the *EventArray* transform (i.e. the 'display' transform) will be applied to the data.

**Returns** a structured array of events in the specified index range.

**arrayValues**(*start=None*, *end=None*, *step=1*, *subchannels=True*, *display=False*)

Get all values in the given index range (w/o times).

**Parameters**

- **start** – The first index in the range, or a slice.

- **end** – The last index in the range. Not used if *start* is a slice.
- **step** – The step increment. Not used if *start* is a slice.
- **subchannels** – A list of subchannel IDs or Boolean. *True* will return all subchannels in native order.
- **display** – If *True*, the *EventArray* transform (i.e. the 'display' transform) will be applied to the data.

**Returns** a structured array of values in the specified index range.

**copy**(*newParent=None*)
Create a shallow copy of the event list.

**exportCsv**(*stream*, *start=None*, *stop=None*, *step=1*, *subchannels=True*, *callback=None*, *callbackInterval=0.01*, *timeScalar=1*, *raiseExceptions=False*, *dataFormat='%.6f'*, *delimiter=', '*, *useUtcTime=False*, *useIsoFormat=False*, *headers=False*, *removeMean=None*, *meanSpan=None*, *display=False*, *noBivariates=False*)
Export events as CSV to a stream (e.g. a file).

**Parameters**

- **stream** – The stream object to which to write CSV data.
- **start** – The first event index to export.
- **stop** – The last event index to export.
- **step** – The number of events between exported lines.
- **subchannels** – A sequence of individual subchannel numbers to export. Only applicable to objects with subchannels. *True* (default) exports them all.
- **callback** – A function (or function-like object) to notify as work is done. It should take four keyword arguments: *count* (the current line number), *total* (the total number of lines), *error* (an exception, if raised during the export), and *done* (will be *True* when the export is complete). If the callback object has a *cancelled* attribute that is *True*, the CSV export will be aborted. The default callback is *None* (nothing will be notified).
- **callbackInterval** – The frequency of update, as a normalized percent of the total lines to export.
- **timeScalar** – A scaling factor for the event times. The default is 1 (microseconds).
- **raiseExceptions** –
- **dataFormat** – The number of decimal places to use for the data. This is the same format as used when formatting floats.
- **useUtcTime** – If *True*, times are written as the UTC timestamp. If *False*, times are relative to the recording.
- **useIsoFormat** – If *True*, the time column is written as the standard ISO date/time string. Only applies if *useUtcTime* is *True*.
- **headers** – If *True*, the first line of the CSV will contain the names of each column.
- **removeMean** – Overrides the EventArray's mean removal for the export.
- **meanSpan** – The span of the mean removal for the export. -1 removes the total mean.
- **display** – If *True*, export using the EventArray's 'display' transform (e.g. unit conversion).

**Returns** Tuple: The number of rows exported and the elapsed time.

getEventIndexBefore(*t*)
Get the index of an event occurring on or immediately before the specified time.

**Parameters** **t** – The time (in microseconds)

**Returns** The index of the event preceding the given time, -1 if the time occurs before the first event.

getEventIndexNear(*t*)
The the event occurring closest to a specific time.

**Parameters** **t** – The time (in microseconds)

**Returns**

getInterval()
Get the first and last event times in the set.

getMax(*startTime=None*, *endTime=None*, *display=False*, *iterator=<built-in function iter>*)
Get the event with the maximum value, optionally within a specified time range. For Channels, returns the maximum among all Subchannels.

**Parameters**

- **startTime** – The starting time. Defaults to the start.
- **endTime** – The ending time. Defaults to the end.
- **display** – If *True*, the final 'display' transform (e.g. unit conversion) will be applied to the results.

**Returns** The event with the maximum value.

getMean(*startTime=None*, *endTime=None*, *display=False*, *iterator=<built-in function iter>*)
Get the mean value of all events, optionally within a specified time range. For Channels, returns the mean among all Subchannels.

**Parameters**

- **startTime** – The starting time. Defaults to the start.
- **endTime** – The ending time. Defaults to the end.
- **display** – If *True*, the final 'display' transform (e.g. unit conversion) will be applied to the results.

**Returns** The event with the minimum value.

getMeanNear(*t*, *outOfRange=False*)
Retrieve the mean value near a given time.

getMin(*startTime=None*, *endTime=None*, *display=False*, *iterator=<built-in function iter>*)
Get the event with the minimum value, optionally within a specified time range. For Channels, returns the minimum among all Subchannels.

**Parameters**

- **startTime** – The starting time. Defaults to the start.
- **endTime** – The ending time. Defaults to the end.
- **display** – If *True*, the final 'display' transform (e.g. unit conversion) will be applied to the results.

**Returns** The event with the minimum value.

getMinMeanMax(*startTime=None*, *endTime=None*, *padding=0*, *times=True*, *display=False*, *iterator=<built-in function iter>*)

Get the minimum, mean, and maximum values for blocks within a specified interval. (Currently an alias of *arrayMinMeanMax*.)

**Todo** Remember what *padding* was for, and either implement or remove it completely. Related to plotting; see *plots*.

**Parameters**

- **startTime** – The first time (in microseconds by default), *None* to start at the beginning of the session.
- **endTime** – The second time, or *None* to use the end of the session.
- **times** – If *True* (default), the results include the block's starting time.
- **display** – If *True*, the final 'display' transform (e.g. unit conversion) will be applied to the results.

**Returns** A structured array of data block statistics (min, mean, and max, respectively).

getRange(*startTime=None*, *endTime=None*, *display=False*)

Get a set of data occurring in a given time interval. (Currently an alias of *arrayRange*.)

**Parameters**

- **startTime** – The first time (in microseconds by default), *None* to start at the beginning of the session.
- **endTime** – The second time, or *None* to use the end of the session.

**Returns** a collection of events in the specified time interval.

getRangeIndices(*startTime*, *endTime*)

Get the first and last event indices that fall within the specified interval.

**Parameters**

- **startTime** – The first time (in microseconds by default), *None* to start at the beginning of the session.
- **endTime** – The second time, or *None* to use the end of the session.

getRangeMinMeanMax(*startTime=None*, *endTime=None*, *subchannel=None*, *display=False*, *iterator=<built-in function iter>*)

Get the single minimum, mean, and maximum value for blocks within a specified interval. Note: Using this with a parent channel without specifying a subchannel number can produce meaningless data if the channels use different units or are on different scales.

**Parameters**

- **startTime** – The first time (in microseconds by default), *None* to start at the beginning of the session.
- **endTime** – The second time, or *None* to use the end of the session.
- **subchannel** – The subchannel ID to retrieve, if the EventArray's parent has subchannels.
- **display** – If *True*, the final 'display' transform (e.g. unit conversion) will be applied to the results.

**Returns** A namedtuple of aggregated event statistics (min, mean, and max, respectively).

`getSampleRate`(*idx=None*)
Get the channel's sample rate. This is either supplied as part of the channel definition or calculated from the actual data and cached.

**Parameters** `idx` – Because it is possible for sample rates to vary within a channel, an event index can be specified; the sample rate for that event and its siblings will be returned.

**Returns** The sample rate, as samples per second (float)

`getSampleTime`(*idx=None*)
Get the time between samples.

**Parameters** `idx` – Because it is possible for sample rates to vary within a channel, an event index can be specified; the time between samples for that event and its siblings will be returned.

**Returns** The time between samples (us)

`getTransforms`(*id_=None*, *_tlist=None*)
Get a list of all transforms applied to the data, from first (the lowest-level parent) to last (the transform, if any, on the object itself).

`getValueAt`(*at*, *outOfRange=False*, *display=False*)
Retrieve the value at a specific time, interpolating between existing events.

**Todo** Optimize. This creates a bottleneck in the calibration.

**Parameters**

- `at` – The time at which to take the sample.
- `outOfRange` – If *False*, times before the first sample or after the last will raise an *IndexError*. If *True*, the first or last time, respectively, is returned.

`hierarchy`()
Get a list of parents/grandparents all the way back to the root. The root is the first item in the list.

`iterJitterySlice`(*start=None*, *end=None*, *step=1*, *jitter=0.5*, *display=False*)
Create an iterator producing events for a range of indices.

**Parameters**

- `start` – The first index in the range, or a slice.
- `end` – The last index in the range. Not used if *start* is a slice.
- `step` – The step increment. Not used if *start* is a slice.
- `jitter` – The amount by which to vary the sample time, as a normalized percentage of the regular time between samples.
- `display` – If *True*, the *EventArray* transform (i.e. the 'display' transform) will be applied to the data.

**Returns** an iterable of events in the specified index range.

`iterMinMeanMax`(*startTime=None*, *endTime=None*, *padding=0*, *times=True*, *display=False*)
Get the minimum, mean, and maximum values for blocks within a specified interval.

**Todo** Remember what *padding* was for, and either implement or remove it completely. Related to plotting; see *plots*.

**Parameters**

- `startTime` – The first time (in microseconds by default), *None* to start at the beginning of the session.

- **endTime** – The second time, or *None* to use the end of the session.
- **times** – If *True* (default), the results include the block's starting time.
- **display** – If *True*, the final 'display' transform (e.g. unit conversion) will be applied to the results.

**Returns** An iterator producing sets of three events (min, mean, and max, respectively).

**iterRange**(*startTime=None*, *endTime=None*, *step=1*, *display=False*)
Get a set of data occurring in a given interval.

**Parameters**

- **startTime** – The first time (in microseconds by default), *None* to start at the beginning of the session.
- **endTime** – The second time, or *None* to use the end of the session.

**iterResampledRange**(*startTime*, *stopTime*, *maxPoints*, *padding=0*, *jitter=0*, *display=False*)
Retrieve the events occurring within a given interval, undersampled as to not exceed a given length (e.g. the size of the data viewer's screen width).

**Todo** Optimize iterResampledRange(); not very efficient, particularly not with single-sample blocks.

**iterSlice**(*start=None*, *end=None*, *step=1*, *display=False*)
Create an iterator producing events for a range of indices.

**Parameters**

- **start** – The first index in the range, or a slice.
- **end** – The last index in the range. Not used if *start* is a slice.
- **step** – The step increment. Not used if *start* is a slice.
- **display** – If *True*, the *EventArray* transform (i.e. the 'display' transform) will be applied to the data.

**Returns** an iterable of events in the specified index range.

**itervalues**(*start=None*, *end=None*, *step=1*, *subchannels=True*, *display=False*)
Iterate all values in the given index range (w/o times).

**Parameters**

- **start** – The first index in the range, or a slice.
- **end** – The last index in the range. Not used if *start* is a slice.
- **step** – The step increment. Not used if *start* is a slice.
- **subchannels** – A list of subchannel IDs or Boolean. *True* will return all subchannels in native order.
- **display** – If *True*, the *EventArray* transform (i.e. the 'display' transform) will be applied to the data.

**Returns** an iterable of structured array value blocks in the specified index range.

**path**()
Get the combined names of all the object's parents/grandparents.

**setTransform**(*transform*, *update=True*)
Set the transforming function/object. This does not change the value of *raw*, however; the new transform will not be applied unless it is *True*.

**updateTransforms**(*recurse=True*)
(Re-)Build and (re-)apply the transformation functions.

**class** idelib.dataset.**WarningRange**(*dataset*, *warningId=None*, *channelId=None*, *subchannelId=None*, *low=None*, *high=None*, *attributes=None*)
An object for indicating when a set of events goes outside of a given range. Originally created for flagging periods of extreme temperatures that will affect accelerometer readings.

For efficiency, the source data should have relatively few samples (e.g. a low sample rate).

Constructor.

**property displayName**
A nice, human-readable description of this warning range, for use with user interfaces.

**getRange**(*start=None*, *end=None*, *sessionId=None*, *iterator=<built-in function iter>*)
Retrieve the invalid periods within a given range of events.

**Returns** A list of invalid periods' [start, end] times.

**getSessionSource**(*sessionId=None*)

**getValueAt**(*at*, *sessionId=None*, *source=None*)
Retrieve the value at a specific time.

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## i

# INDEX

## I



## L



## M



## P



## S



## U



## W