PACFISH

Members of the International Photoacoustic Standardisation Cons

Feb 24, 2023

GENERAL INFORMATION

Welcome to the PACFISH documentation!

CHAPTER

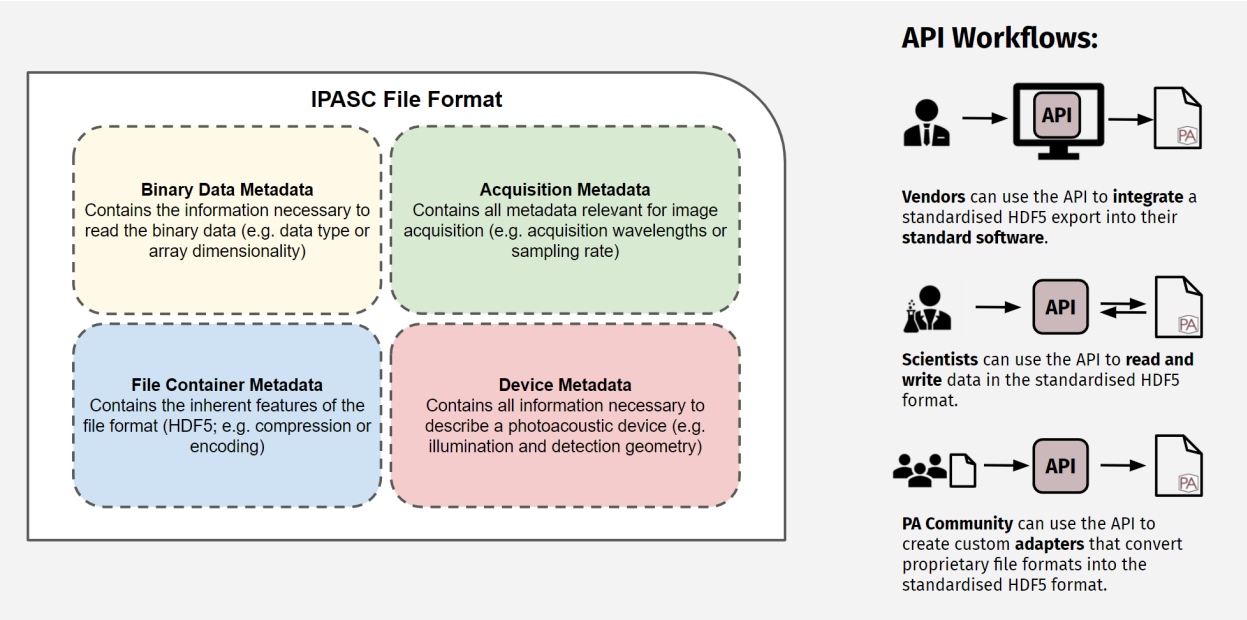
ONE

PACFISH

In this repository we develop the photoacoustic converter for information sharing (PACFISH). It is a tool that enables the conversion of vendor-specific proprietary data formats into the IPASC data format, which is an HDF5 container that has a defined structure for the meta data that are given with the binary data. A list of meta data information was suggested by the International Photoacoustic Standardisation Consortium (IPASC) in early 2020. You can find this list using the following link:

https://www.ipasc.science/documents/20210916_IPASC_Format_V2.pdf

PACFISH serves three purposes: (1) it helps vendors to integrate the IPASC data format export into their standard software; (2) it assists scientists to read and write data in the consensus HDF5 format; and (3) it helps the PA community to create custom adapters that convert proprietary file formats into the consensus HDF5 format.

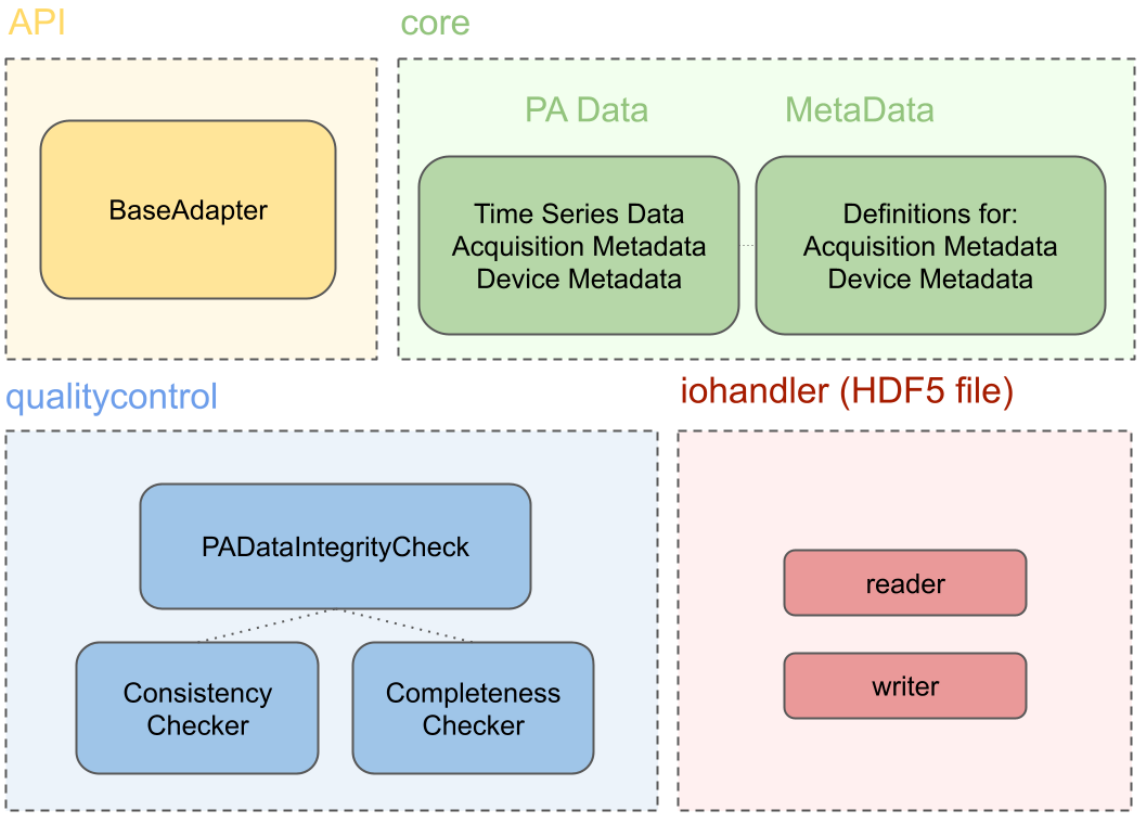


Please help IPASC by reporting any missing parameters, bugs, or issues. We are also looking forward to any contributions in form of adapters that can convert a priprietary format into the IPASC format. If you are a member of the research community, a photoacoustic vendor, or interested to contribute or in the project in general because of any other reasons, please contact the leadership team of the Data Acquisition and Management Theme of IPASC.

These are currently: Janek Gröhl, Lina Hacker, and Ben Cox.

1.1 Software Architecture

PACFISH is divided into the API, core, quality control, and iohandler modules. The API package (*pacfish.api* yellow module) can be used to facilitate the integration of conversion adapters to convert from arbitrary file formats into the IPASC data format. To create a conversion adapter, a Python representation of (1) the binary data, (2) the acquisition metadata dictionary, and (3) the device metadata dictionary need to be implemented.



The core classes (*pacfish.core* green module) represent the metadata and data structure in Python. Each metadatum is described with specific device tags defining the name, data type, necessity and SI unit (if applicable), and setting a value constraint. Basic metadata constraints have been implemented to avoid accidental typos within the values field (e.g. only positive numbers larger than zero are applicable for acquisition wavelengths). If the value is not within the constraints a *TypeError* is raised. Metadatum-specific functions enable easy addition of the values for the specific metadata field.

The quality control functionalities (*pacfish.qualitycontrol* blue module) ensure the correctness of the conversion into the IPASC format: a *completeness checker* tests that all metadata are being called and a *consistency checker* ensures that all metadata are within their constraints. An automatically-generated output report gives a human-readable summary of the quality control checks and ensures that the likelihood of conversion mistakes are minimized. For control of the *Device Metadata*, the detector and illuminator positions can be represented in a 3D coordinate system as visual control.

Finally, the I/O functionality (*pacfish.iohandler* red module) enables reading and writing of IPASC-formatted data files.

CHAPTER

TWO

EXAMPLES AND USE CASES

Please look in the `examples` folder for detailed and functional examples how to use the PACFISH API. We have examples for both `Python` and `MATLAB`.

2.1 Use case: using the tool to read and write HDF5 files

```
import pacfish as pf

# Loading data from the hard drive
pa_data = pf.load_data("path/to/hdf5file.hdf5")
numpy_array = pa_data.binary_time_series_data

# Writing of data to hard drive
pf.write_data("path/to/new/file.hdf5", pa_data)
```

2.2 Use case: Implement a conversion adapter

```
impot pacfish as pf

class DeviceSpecificAdapter(pf.BaseAdapter):

    def generate_binary_data(self) -> np.ndarray:
        # IMPLEMENTATION HERE
        pass

    def generate_device_meta_data(self) -> dict:
        # IMPLEMENTATION HERE
        pass

    def set_metadata_value(self, metadata_tag: MetaDatum) -> object:
        # IMPLEMENTATION HERE
        pass
```

CHAPTER

THREE

LIST OF CONTRIBUTORS

The following is an alphabetically sorted list of people that have contributed to this project.

Name	Institution	Contributions
Ben Cox	University College London	MATLAB API
Kris Dreher	German Cancer Research Center	IO Handling
Janek Gröhl	University of Cambridge	General Maintenance; Python API
Lina Hacker	University of Cambridge	Python API
François Varray	Creatis, Université de Lyon	MATLAB API
Jeffrey Sackey	University College London	MATLAB API
Lawrence Yip	Lawson Health Research Institute	Adapter Implementation, Bug Fixing

CHAPTER

FOUR

HOW TO CONTRIBUTE

We welcome any forms of contributions to the project! If you are unsure how to contribute, contact either Ben Cox, Lina Hacker, or Janek Gröhl (@jgroehl) for guidance. Before contributing you should be aware of some boundary conditions that are outlined here:

4.1 License

The project is licensed under the BSD 3-Clause License. Every contributor has to make their contributions available under the same license. Including materials from other licenses is only possible, if the respective license is compatible with the BSD 3-Clause License.

4.2 Copyright

Every contributor (or their institution) will retain their copyright. The copyrights applicable to each file in the code will be made clear explicitly using the SPDX standard in a file header:

```
"""
SPDX-FileCopyrightText: 2021 Random Author Name
SPDX-License-Identifier: BSD 3-Clause License
"""
```

By contributing, authors have to have the rights to actually contribute the code to the project and agree to the developer's certificate of origin:

4.3 Developer's Certificate of Origin

When contributing to the project, you agree to the following terms, stating that you have indeed the right to contribute the code under the MIT license and that you acknowledge that the contributed code will be and remain publically available.

```
By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I
    have the right to submit it under the open source license
    indicated in the file; or

(b) The contribution is based upon previous work that, to the best
```

(continues on next page)

(continued from previous page)

```
    of my knowledge, is covered under an appropriate open source
    license and I have the right under that license to submit that
    work with modifications, whether created in whole or in part
    by me, under the same open source license (unless I am
    permitted to submit under a different license), as indicated
    in the file; or

(c) The contribution was provided directly to me by some other
    person who certified (a), (b) or (c) and I have not modified
    it.

(d) I understand and agree that this project and the contribution
    are public and that a record of the contribution (including all
    personal information I submit with it, including my sign-off) is
    maintained indefinitely and may be redistributed consistent with
    this project or the open source license(s) involved.
```

To validate that you agree with these terms, please sign off the last commit before your pull request, by adding the following line to the commit message:

```
Signed-off-by: Random J Developer <random@developer.example.org>
```

This is a built-in feature of git and you can automate this by using the `-s` flag.

4.4 Coding conventions

We ask all contributors to follow a couple of conventions and best pratices when contributing code:

- Code is formatted according to the Python PEP-8 coding standard.
- Contributors create a test case that tests their code.
- Contributors document their code.

4.5 Practical Workflow

Contributors open issue and create implementation on a separate branch or fork. Any open questions / calls for help are addressed via a meeting taking place every second week or the comment function in the issue. Once the contributor is happy with their code they sign-off the last commit and open a pull request.

CHAPTER

FIVE

PACFISH

5.1 api

5.1.1 adapters

This package contains implementations of custom adapters that enable the conversion of certain commonly used photoacoustic data formats into the IPASC format.

You can use these as a reference when attempting to define your own adapters.

class NrrdFileConverter(*nrrd_file_path*)

Bases: *BaseAdapter*

This converter assumes a linear transducer with 128 elements and an element pitch of 0.3mm. It assumes that the NRRD file metadata contains a 'sizes', 'type' and 'space directions' field.

generate_binary_data() → ndarray

The binary data is the raw time series data. It is internally stored as an N-dimensional numpy array. The binary data must be formatted the following way:

[detectors, samples, wavelengths, measurements]

Returns
A numpy array containing the binary data

Return type
np.ndarray

generate_device_meta_data() → dict

Must be implemented to define a digital twin of the photoacoustic imaging device. This method can be implemented using the DeviceMetaDataCreator.

Returns
A dictionary containing all key-value pair necessary to describe a digital twin of a photoacoustic device.

Return type
dict

set_metadata_value(*metadatum:* MetaDatum) → object

This method must be implemented to yield appropriate data for all MetaDatum elements in the MetaDataTags class.

You are given a certain meta datum nd have to return the appropriate information.

Parameters
metadatum (MetaDatum) – The MetaDatum for which to return the correct data.

Returns
The data corresponding to the given MetaDatum

Return type
object

class BaseAdapter

Bases: ABC

The purpose of the BaseAdapter class is to provide the framework to convert from any given input data type into the IPASC format. It can be used as a basis for extension for a custom Adapter.

To achieve this, one needs to inherit from BaseAdapter and implement the abstract methods:

```
class CustomAdapter(BaseAdapter):

    def __init__():
        # TODO do all of the loading etc here
        # Then call the __init__ of the BaseAdapter
        super(CustomAdapter, self).__init__()
        # TODO Add custom parameters after calling BaseAdapter.__init__

    def generate_binary_data(self):
        # TODO

    def generate_device_meta_data(self):
        # TODO

    def set_metadata_value(self, metadatum: MetaDatum):
        # TODO
```

add_custom_meta_datum_field(*key: str*, *value: object*)

This method can be used to add a metadata field that is not reflected in the standard list of metadata of the IPASC format. Must be called after the __init__ method of the BaseAdapter was called. The custom meta data are stored in the AcquisitionMetadata dictionary.

Parameters

- **key** (*str*) – The unique name with which the value is stored in the dictionary.
- **value** (*object*) – The value to store.

generate_acquisition_meta_data() → dict

Internal method

Return type
dict

abstract generate_binary_data() → ndarray

The binary data is the raw time series data. It is internally stored as an N-dimensional numpy array. The binary data must be formatted the following way:

[detectors, samples, wavelengths, measurements]

Returns
A numpy array containing the binary data

Return type
np.ndarray

abstract generate_device_meta_data() → dict

Must be implemented to define a digital twin of the photoacoustic imaging device. This method can be implemented using the DeviceMetaDataCreator.

Returns
A dictionary containing all key-value pair necessary to describe a digital twin of a photoacoustic device.

Return type
dict

generate_pa_data() → *PAData*

Internal method

Return type
PAData

abstract set_metadata_value(*metadatum:* MetaDatum) → object

This method must be implemented to yield appropriate data for all MetaDatum elements in the MetadataTags class.

You are given a certain meta datum nd have to return the appropriate information.

Parameters
metadatum (MetaDatum) – The MetaDatum for which to return the correct data.

Returns
The data corresponding to the given MetaDatum

Return type
object

5.2 core

class DetectionElementCreator

Bases: object

A DetectionElementCreator can be used to create detection elements for the purposes of a standardised device representation within the IPASC data format.

It should be used in the following way:

```
dec = DetectionElementCreator()
dec.set_detector_position(position)
# ... set other attributes
element = dec.get_dictionary()
```

The *element* dictionary can then be added to the *DeviceMetaDataCreator*.

get_dictionary()

Returns a copy of a dictionary describing the created detection element up to this point. Subsequent changes to the element via the *DetectionElementCreator* will **not** alter the dictionary returned by this function. If changes are done this functions needs to be called again.

Returns
A dictionary representing the created detection element.

Return type
dict

`set_angular_response`(*angular_response: ndarray*)

Parameters
angular_response – a two element array [angles, response] describing the angular response of the detecor. Angles and response are also arrays where len(angles) == len(response). The units can be found in MetadataDeviceTags.ANGULAR_RESPONSE.unit.

Returns
No return value

Return type
None

`set_detector_geometry`(*geometry*)

Parameters
geometry – a three element array [x1, x2, x3] describing the extent of the detector size in x1, x2, and x3 direction. The units can be found in MetadataDeviceTags.DETECTOR_SIZE.unit.

Returns
No return value

Return type
None

`set_detector_geometry_type`(*geometry_type: str*)

Parameters
geometry_type – The detector geometry type defines how to interpret the data in the detector geometry field. The following geometry types are currently supported:

- "CIRCULAR" - defined by a single value that determines the radius of the circle
- "SPHERE" - defined by a single value that determines the radius of the sphere
- "CUBOID" - defined by three values that determine the extent of the cuboid in x, y, and z dimensions, before the position and orientation transforms.
- "MESH" - defined by a STL-formatted string that determines the positions of points and faces before the position and orientation transforms.

Returns
No return value

Return type
None

`set_detector_orientation`(*orientation: ndarray*)

Parameters
orientation – a n array of three float values that describe the orientation of the detector element in the x1, x2, and x3 direction. The units can be found in MetadataDeviceTags.DETECTOR_ORIENTATION.unit.

Returns
No return value

Return type
None

set_detector_position(*detector_position: ndarray*)

Parameters
detector_position – an array of three float values that describe the position of the detection element in the x1, x2, and x3 direction. The units can be found in MetadataDeviceTags.DETECTOR_POSITION.unit.

Returns
No return value

Return type
None

set_frequency_response(*frequency_response: ndarray*)

Parameters
frequency_response – a two element array [frequency, response] describing the frequency response of the detector. Frequency and response are also arrays where len(frequency) == len(response). The units can be found in MetadataDeviceTags.FREQUENCY_RESPONSE.unit.

Returns
No return value

Return type
None

class DeviceMetaDataCreator

Bases: `object`

A helper class to create a dictionary that describes a digital device twin according to the IPASC data format. In the interplay with the *DetectionElementCreator* and the *IlluminationElementCreator*, elements can be added to the representation.

Example:

```
dmdc = DeviceMetaDataCreator()
dmdc.set_general_information(uuid, fov)
for _ in range(num_detection_elements):
    dec = DetectionElementCreator()
    dec.set_detector_position(position)
    # ... set other attributes
    element = dec.get_dictionary()
    dmdc.add_detection_element(element)
for _ in range(num_illuminators):
    iec = IlluminationElementCreator()
    iec.set_illuminator_position(position)
    # ... set other attributes
    element = iec.get_dictionary()
    dmdc.add_detection_element(element)
device_metadata_dict = dmdc.finalize_device_meta_data()
```

Initialises the DeviceMetaDataCreator.

add_detection_element(*detection_element: dict*)

Parameters
detection_element – is a dictionary for the detection element specific parameters

Returns
No return value

Return type
None

add_illumination_element(*illumination_element: dict*)

Parameters
illumination_element – is a dictionary for the illumination element specific parameters

Returns
No return value

Return type
None

finalize_device_meta_data()

Returns a copy of a dictionary describing the created device up to this point. Subsequent changes to the element via the *DeviceMetaDataCreator* will **not** alter the dictionary returned by this function. If changes are done this functions needs to be called again.

Returns
A dictionary representing the created digital device twin.

Return type
dict

set_general_information(*uuid: str*, *fov: ndarray*)

Parameters

- **uuid** – is a string that uniquely identifies the photoacoustic device
- **fov** – is an array of six float values that describe the extent of the field of view of the device in the x1, x2, and x3 directions: [x1_start, x1_end, x2_start, x2_end, x3_start, x3_end].

Returns
No return value

Return type
None

class IlluminationElementCreator

Bases: object

A IlluminationElementCreator can be used to create illumination elements for the purposes of a standardised device representation within the IPASC data format.

It should be used in the following way:

```
iec = IlluminationElementCreator()
iec.set_illuminator_position(position)
# ... set other attributes
element = iec.get_dictionary()
```

The *element* dictionary can then be added to the *DeviceMetaDataCreator*.

Instantiates a IlluminationElementCreator.

`get_dictionary`()

Returns a copy of a dictionary describing the created illumination element up to this point. Subsequent changes to the element via the *IlluminationElementCreator* will **not** alter the dictionary returned by this function. If changes are done this functions needs to be called again.

Returns
A dictionary representing the created illumination element.

Return type
dict

`set_beam_divergence_angles`(*angle: float*)

Parameters
angle – a value describing the opening angle of the laser beam from the illuminator shape with respect to the orientation vector. This angle is represented by the standard deviation of the beam divergence. The units can be found in MetadataDeviceTags.BEAM_DIVERGENCE_ANGLES.unit.

Returns
No return value

Return type
None

`set_beam_energy_profile`(*energy_profile: ndarray*)

Parameters
energy_profile – a two element array [wavelengths, laser_energy] describing the laser energy profile. beam energy and wavelengths are also arrays where len(laser_energy) == len(profile) The units can be found in MetadataDeviceTags.BEAM_ENERGY_PROFILE.unit.

Return type
None

`set_beam_intensity_profile`(*intensity_profile: ndarray*)

Parameters
intensity_profile – a two element array [wavelengths, intensity_profile] describing the beam itensity profile. Wavelengths and intensity_profile are also arrays where len(wavelengths) == len(intensity_profile) The units can be found in MetadataDeviceTags.BEAM_INTENSITY_PROFILE.unit.

Return type
None

`set_beam_stability_profile`(*stability_profile: ndarray*)

Parameters
stability_profile – a two element array [wavelengths,laser_stability,] describing the laser stability profile. Beam stability and wavelengths are also arrays where len(stability_profile) == len(wavelengths). The units can be found in MetadataDeviceTags.BEAM_STABILITY_PROFILE.unit.

Return type
None

`set_illuminator_geometry`(*shape: ndarray*)

Parameters
shape – is an array of three float values that describe the shape of the illuminator in the x1, x2, and x3 direction. The units can be found in MetadataDeviceTags.ILLUMINATOR_GEOMETRY.unit.

Return type
None

set_illuminator_geometry_type(*illuminator_geometry_type: str*)

Parameters
illuminator_geometry_type – The illuminator geometry type defines how to interpret the data in the illuminator geometry field. The following geometry types are currently supported:

- "CIRCULAR" - defined by a single value that determines the radius of the circle
- "SPHERE" - defined by a single value that determines the radius of the sphere
- "CUBOID" - defined by three values that determine the extent of the cuboid in x, y, and z dimensions, before the position and orientation transforms.
- "MESH" - defined by a STL-formatted string that determines the positions of points and faces before the position and orientation transforms.

Return type
None

set_illuminator_orientation(*orientation: ndarray*)

Parameters
orientation – is an array of three float values that describe the orientation of the illumination element in the x1, x2, and x3 direction. The units can be found in MetadataDeviceTags.ILLUMINATOR_ORIENTATION.unit.

Return type
None

set_illuminator_position(*illuminator_position: ndarray*)

Parameters
illuminator_position – is an array of three float values that describe the position of the illumination element in the x1, x2, and x3 direction. The units can be found in MetadataDeviceTags.ILLUMINATOR_POSITION.unit.

Return type
None

set_pulse_width(*pulse_width: float*)

Parameters
pulse_width – a floating point value describing the pulse width of the laser in the units of MetadataDeviceTags.PULSE_WIDTH.unit.

Return type
None

set_wavelength_range(*wl_range: ndarray*)

Parameters
wl_range – is an array of three float values that describe the minimum wavelength lambda_min, the maximum wavelength lambda_max and a metric for

the accuracy lambda_accuracy. The units can be found in MetadataDeviceTags.WAVELENGTH_RANGE.unit.

Return type
None

`DIMENSIONALITY_STRINGS = ['time', 'space', 'time and space']`

The Dimenstionality_STRINGS define what the value space of the metadatum DIMENSIONALITY is.

`class EnumeratedString(tag, minimal, dtype, unit='N/A', permissible_strings=None)`

Bases: *MetaDatum*

This MetaDatum is defined to be a string that must be from a defined list of strings.

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".
- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.
- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises
TypeError: – if one of the parameters is not of the correct type.

`evaluate_value_range(value) → bool`

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters
value (*object*) – value to evaluate

Returns
True if the given value is acceptable for the respective MetaDatum

Return type
bool

`class MetaDatum(tag: str, minimal: bool, dtype: (<class 'type'>, <class 'tuple'>), unit: str = 'N/A')`

Bases: ABC

This class represents a meta datum. A meta datum contains all necessary information to fully characterize the meta information represented by an instance of this class.

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".
- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.

- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises

TypeError: – if one of the parameters is not of the correct type.

abstract evaluate_value_range(*value*) → bool

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters

value (*object*) – value to evaluate

Returns

True if the given value is acceptable for the respective MetaDatum

Return type

bool

class MetadataAcquisitionTags

Bases: object

This class defines the MetaData that compose all information needed to describe the measurement circumstances for a given measurement of photoacoustic time series data.

It also specifies the naming conventions of the underlying HDF5 data fields. Furthermore, it is specified if a certain meta datum is minimal or not, the data type is defined and the units of the metadatum are given.

ACOUSTIC_COUPLING_AGENT = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

ACQUISITION_WAVELENGTHS = <pacfish.core.Metadata.NDimensionalNumpyArray object>

AD_SAMPLING_RATE = <pacfish.core.Metadata.NonNegativeNumber object>

COMPRESSION = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

DATA_TYPE = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

DIMENSIONALITY = <pacfish.core.Metadata.EnumeratedString object>

ELEMENT_DEPENDENT_GAIN = <pacfish.core.Metadata.NonNegativeNumbersInArray object>

ENCODING = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

FREQUENCY_DOMAIN_FILTER = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

MEASUREMENTS_PER_IMAGE = <pacfish.core.Metadata.NonNegativeWholeNumber object>

MEASUREMENT_SPATIAL_POSES = <pacfish.core.Metadata.NDimensionalNumpyArray object>

MEASUREMENT_TIMESTAMPS = <pacfish.core.Metadata.NonNegativeNumbersInArray object>

OVERALL_GAIN = <pacfish.core.Metadata.NonNegativeNumber object>

PHOTOACOUSTIC_IMAGING_DEVICE_REFERENCE =
<pacfish.core.Metadata.UnconstrainedMetaDatum object>

PULSE_ENERGY = <pacfish.core.Metadata.NonNegativeNumbersInArray object>

REGIONS_OF_INTEREST = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

SCANNING_METHOD = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

SIZES = <pacfish.core.Metadata.NonNegativeNumbersInArray object>

SPEED_OF_SOUND = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

TAGS = [<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.EnumeratedString object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NonNegativeNumber object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeNumber object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeWholeNumber object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>]

TAGS_ACQUISITION = [<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NonNegativeNumber object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeNumber object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeWholeNumber object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>]

TAGS_BINARY = [<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.EnumeratedString object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>]

TAGS_CONTAINER = [<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>]

TEMPERATURE_CONTROL = <pacfish.core.Metadata.NonNegativeNumbersInArray object>

TIME_GAIN_COMPENSATION = <pacfish.core.Metadata.NonNegativeNumbersInArray object>

UUID = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

class MetadataDeviceTags

Bases: object

This class defines the MetaData that compose all information needed to describe a digital twin of a photoacoustic device.

It also specifies the naming conventions of the underlying HDF5 data fields. Furthermore, it is specified if a certain meta datum is minimal or not, the data type is defined and the units of the metadatum are given.

ANGULAR_RESPONSE = <pacfish.core.Metadata.NDimensionalNumpyArray object>

BEAM_DIVERGENCE_ANGLES = <pacfish.core.Metadata.NumberWithUpperAndLowerLimit object>

BEAM_ENERGY_PROFILE = <pacfish.core.Metadata.NDimensionalNumpyArray object>

BEAM_INTENSITY_PROFILE = <pacfish.core.Metadata.NDimensionalNumpyArray object>

BEAM_STABILITY_PROFILE = <pacfish.core.Metadata.NDimensionalNumpyArray object>

DETECTION_ELEMENT = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

DETECTORS = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

DETECTOR_GEOMETRY = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

DETECTOR_GEOMETRY_TYPE = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

DETECTOR_ORIENTATION = <pacfish.core.Metadata.NDimensionalNumpyArray object>

DETECTOR_POSITION = <pacfish.core.Metadata.NDimensionalNumpyArray object>

FIELD_OF_VIEW = <pacfish.core.Metadata.NDimensionalNumpyArrayWithMElements object>

FREQUENCY_RESPONSE = <pacfish.core.Metadata.NonNegativeNumbersInArray object>

GENERAL = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

ILLUMINATION_ELEMENT = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

ILLUMINATORS = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

ILLUMINATOR_GEOMETRY = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

ILLUMINATOR_GEOMETRY_TYPE = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

ILLUMINATOR_ORIENTATION = <pacfish.core.Metadata.NDimensionalNumpyArray object>

ILLUMINATOR_POSITION = <pacfish.core.Metadata.NDimensionalNumpyArray object>

INTENSITY_PROFILE_DISTANCE = <pacfish.core.Metadata.NonNegativeNumber object>

NUMBER_OF_DETECTION_ELEMENTS = <pacfish.core.Metadata.NonNegativeWholeNumber object>

NUMBER_OF_ILLUMINATION_ELEMENTS = <pacfish.core.Metadata.NonNegativeWholeNumber object>

PULSE_WIDTH = <pacfish.core.Metadata.NonNegativeNumber object>

TAGS = [<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NDimensionalNumpyArrayWithMElements object>,
<pacfish.core.Metadata.NonNegativeWholeNumber object>,
<pacfish.core.Metadata.NonNegativeWholeNumber object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NonNegativeNumber object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NonNegativeNumber object>,
<pacfish.core.Metadata.NumberWithUpperAndLowerLimit object>]

TAGS_DETECTORS = [<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NonNegativeNumbersInArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>]

TAGS_GENERAL = [<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NDimensionalNumpyArrayWithMElements object>,
<pacfish.core.Metadata.NonNegativeWholeNumber object>,
<pacfish.core.Metadata.NonNegativeWholeNumber object>]

TAGS_ILLUMINATORS = [<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.UnconstrainedMetaDatum object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NonNegativeNumber object>,
<pacfish.core.Metadata.NDimensionalNumpyArray object>,
<pacfish.core.Metadata.NonNegativeNumber object>,
<pacfish.core.Metadata.NumberWithUpperAndLowerLimit object>]

UNIQUE_IDENTIFIER = <pacfish.core.Metadata.UnconstrainedMetaDatum object>

WAVELENGTH_RANGE = <pacfish.core.Metadata.NDimensionalNumpyArray object>

class NDimensionalNumpyArray(*tag*, *minimal*, *dtype*, *unit='N/A'*, *expected_array_dimension=1*)

Bases: *MetaDatum*

This MetaDatum is defined to be an array of unconstrained numbers.

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".
- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.
- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises

TypeError: – if one of the parameters is not of the correct type.

evaluate_value_range(*value*) → bool

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters

value (*object*) – value to evaluate

Returns

True if the given value is acceptable for the respective MetaDatum

Return type

bool

class NDimensionalNumpyArrayWithMElements(*tag*, *minimal*, *dtype*, *unit='N/A'*, *expected_array_dimension=1*, *elements_per_dimension=None*)

Bases: *MetaDatum*

This MetaDatum is defined to be an array with a specific dimensionality.

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".
- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.
- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises
TypeError: – if one of the parameters is not of the correct type.

evaluate_value_range(*value*) → bool

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters
value (*object*) – value to evaluate

Returns
True if the given value is acceptable for the respective MetaDatum

Return type
bool

class NonNegativeNumber(*tag*, *minimal*, *dtype*, *unit='N/A'*)

Bases: *MetaDatum*

This MetaDatum is defined to be a non-negative number.

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".
- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.
- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises
TypeError: – if one of the parameters is not of the correct type.

evaluate_value_range(*value*) → bool

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters
value (*object*) – value to evaluate

Returns
True if the given value is acceptable for the respective MetaDatum

Return type
bool

class NonNegativeNumbersInArray(*tag*, *minimal*, *dtype*, *unit='N/A'*)

Bases: *MetaDatum*

This MetaDatum is defined to be an array containing non-negative whole numbers.

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".

- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.
- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises
TypeError: – if one of the parameters is not of the correct type.

evaluate_value_range(*value*) → bool

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters
value (*object*) – value to evaluate

Returns
True if the given value is acceptable for the respective MetaDatum

Return type
bool

class NonNegativeWholeNumber(*tag*, *minimal*, *dtype*, *unit='N/A'*)

Bases: *MetaDatum*

This MetaDatum is defined to be a non-negative whole number.

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".
- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.
- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises
TypeError: – if one of the parameters is not of the correct type.

evaluate_value_range(*value*) → bool

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters
value (*object*) – value to evaluate

Returns
True if the given value is acceptable for the respective MetaDatum

Return type
bool

class NumberWithUpperAndLowerLimit(*tag*, *minimal*, *dtype*, *unit='N/A'*, *lower_limit=-inf*, *upper_limit=inf*)

Bases: *MetaDatum*

This MetaDatum is defined to be a whole number in between a lower and an upper bound (inclusive).

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".
- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.
- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises

TypeError: – if one of the parameters is not of the correct type.

evaluate_value_range(*value*) → bool

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters

value (*object*) – value to evaluate

Returns

True if the given value is acceptable for the respective MetaDatum

Return type

bool

class UnconstrainedMetaDatum(*tag*, *minimal*, *dtype*, *unit='N/A'*)

Bases: *MetaDatum*

This MetaDatum has no limitations on the values associated with it.

Instantiates a MetaDatum and sets all relevant values.

Parameters

- **tag** (*str*) – The tag that corresponds to this meta datum.
- **minimal** (*bool*) – Defines if the metadatum is *minimal* (i.e. if is MUST be reported). Without the minimal parameters, the time series data cannot be reconstructed into an image. All parameters that are not minimal are interpreted as "report if present".
- **dtype** (*type, tuple*) – The data type of the meta datum. Can either be a single type or a tuple of possible types.
- **unit** (*str*) – The unit associated with this metadatum. Must be one of the strings defined in pacfish.Units.

Raises

TypeError: – if one of the parameters is not of the correct type.

evaluate_value_range(*value*) → bool

Evaluates if a given value fits to the acceptable value range of the MetaDatum.

Parameters

value (*object*) – value to evaluate

Returns

True if the given value is acceptable for the respective MetaDatum

Return type

bool

class Units

Bases: object

A list of the SI and compound units that are used in the IPASC format.

DIMENSIONLESS_UNIT = 'one'

HERTZ = 'Hz'

JOULES = 'J'

KELVIN = 'K'

METERS = 'm'

METERS_PER_SECOND = 'm/s'

NO_UNIT = 'N/A'

RADIANS = 'rad'

SECONDS = 's'

class PAData(*binary_time_series_data: Optional[ndarray] = None*, *meta_data_acquisition: Optional[dict] = None*, *meta_data_device: Optional[dict] = None*)

Bases: object

The PAData class is the core class for accessing the information contained in the HDF5 files. Using the *pacfish.load_data* method yields an instance of this class.

It is structured into three main parts:

1. a numpy array containing the binary data
2. a dictionary with the acquisition metadata
3. a dictionary with the device metadata

Furthermore, this class contains convenience methods to access all fields within the HDF5 dictionary, without the necessity to know the internal structure by heart.

Creates an empty instance of the PAData class. To instantiate with a path to an HDF5 file, please use the *pacfish.load_data* method.

Parameters

- **binary_time_series_data** (*np.ndarray*) – a numpy array that must not be None
- **meta_data_acquisition** (*dict*) – If None will be initialized as an empty dictionary.
- **meta_data_device** (*dict*) – If None will be initialized as an empty dictionary.

Returns
An empty PADta instance to be populated

Return type
pacfish.PAData

get_acoustic_coupling_agent()

A string representing the acoustic coupling agent that is used.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_acquisition_meta_datum`(*meta_data_tag:* MetaDatum) → object

This method returns data from the acquisition metadata dictionary

Parameters
meta_data_tag – the MetaDatum instance for which to get the information.

Returns
return value might be None, if the specified metadata tag was not found in the dictionary.

Return type
object

`get_acquisition_wavelengths`()

A 1D array that contains all wavelengths used for the image acquisition.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_angular_response`(*identifier=None*)

The angular response field characterizes the angular sensitivity of the detection element to the incident angle (relative to the elements orientation) of the incoming pressure wave.

Parameters
identifier (*str*) – The ID of a specific detection element. If *None* then all detection elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_beam_divergence`(*identifier=None*)

The beam divergence angles represent the opening angles of the beam from the illuminator shape with respect to the orientation vector. This angle represented by the standard deviation of the beam divergence.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
float

`get_beam_energy_profile`(*identifier=None*)

The beam energy profile field is a discretized functional of wavelength (nm) that represents the light energy of the illuminator with regard to the wavelength. Thereby, systematic differences in multispectral image acquisitions can be accounted for.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_beam_profile`(*identifier=None*)

The beam intensity profile is a function of a spatial position that specifies the relative beam intensity according to the planar emitting surface of the illuminator shape.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_beam_profile_distance`(*identifier=None*)

The distance from the light source for measuring its beam intensity profile.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
float

`get_beam_stability_profile`(*identifier=None*)

The beam noise profile field is a functional of wavelength (nm) that represents the standard deviation of the pulse-to-pulse energy of the illuminator with regard to the wavelength.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarrayy

`get_compression`()

The compression field is representative of the compression method that was used to compress the binary data. E.g. one of 'raw', 'gzip', ...

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_custom_meta_datum`(*meta_data_tag: str*) → object

This method returns data from the acquisition metadata dictionary.

Parameters
meta_data_tag – a string instance for which to get the information.

Returns
return value might be None, if the specified metadata tag was not found in the dictionary.

Return type
object

`get_data_UUID()`

128-bit Integer displayed as a hexadecimal string in 5 groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters. The UUID is randomly generated using the UUID Version 4 standard.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_data_type()`

The data type field represents the datatype of the binary data. This field is given in the C++ data type naming convention. E.g. 'short', 'unsigned short', 'int', 'unsigned int', 'long', 'unsigned long', 'long long', 'float', 'double', 'long double'.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_detector_attribute_for_tag(metadatum, identifier=None)`

Method all convenience functions regarding the detection elements are delegated to.

Parameters

- **metadatum** (MetaDatum) – The metadatum that corresponds to the information that should be extracted from the metadata dictionary.
- **identifier** (*str*) – The ID of a specific detection element. If *None* then all detection elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
object

`get_detector_geometry(identifier=None)`

The element size defines the size of the detection element in 3D cartesian coordinates [x1, x2, x3] relative to its position and orientation.

Parameters
identifier (*str*) – The ID of a specific detection element. If *None* then all detection elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_detector_geometry_type(identifier=None)`

The detector geometry type defines how to interpret the data in the detector geometry field. The following geometry types are currently supported:

- "CIRCULAR" - defined by a single value that determines the radius of the circle
- "SPHERE" - defined by a single value that determines the radius of the sphere
- "CUBOID" - defined by three values that determine the extent of the cuboid in x, y, and z dimensions before the position and orientation transforms.
- "MESH" - defined by a STL-formatted string that determines the positions of points and faces before the position and orientation transforms.

Parameters
identifier (*str*) – The ID of a specific detection element. If *None* then all detection elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

get_detector_ids() → list

Returns a list of all IDs of the detection elements that are added in this PAData instance.

Returns
a list of all ids of the detection elements

Return type
list

get_detector_orientation(*identifier=None*)

The element orientation defines the rotation of the detection element in 3D cartesian coordinates [r1, r2, r3] in radians.

Parameters
identifier (*str*) – The ID of a specific detection element. If *None* then all detection elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

get_detector_position(*identifier=None*)

The detector position defines the position of the detection element centroid in 3D cartesian coordinates [x1, x2, x3].

Parameters
identifier (*str*) – The ID of a specific detection element. If *None* then all detection elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

get_device_uuid()

The UUID is a universally unique identifier to the device description that can be referenced.

Returns
return value can be None, of no UUID was found in the metadata.

Return type
str

`get_dimensionality()`

The dimensionality field represents the acquisition format of the binary data and specifies the number of spatiotemporal dimensions of the data that is comprised of one or more frames. E.g. '1D', '2D', '3D', '1D+t', 2D+t', '3D+t'. In this notion, the time series sampling of one transducer would count as a "spatial" dimension. These are defined as 1D = [], 2D = [x1,], 3D = [x1, , x2]. The "+t" will then add a time dimension for multiple of these frames.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_element_dependent_gain()`

An array that contains the relative factors used for apodisation or detection element-wise sensitivity corrections.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_encoding()`

The encoding field is representative of the character set that was used to encode the binary data and the metadata. E.g. one of 'UTF-8', 'ASCII', 'CP-1252', ...

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_field_of_view()`

An array defining an approximate cuboid (3D) area that should be reconstructed in 3D Cartesian coordinates [x1_start, x1_end, x2_start, x2_end, x3_start, x3_end]. A 2D Field of View can be defined by setting the start and end coordinate of the respective dimension to the same value.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_frequency_domain_filter()`

The frequency threshold levels that have been applied to filter the raw time series data.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_frequency_response(`*identifier=None*`)`

The frequency response is a functional that characterizes the response of the detection element to the frequency of the incident pressure waves.

Parameters
identifier (*str*) – The ID of a specific detection element. If *None* then all detection elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

get_illuminator_attribute_for_tag(*metadatum*, *identifier=None*)

Method all convenience functions regarding the illumination elements are delegated to.

Parameters

- **metadatum** (*MetaDatum*) – The metadatum that corresponds to the information that should be extracted from the metadata dictionary.
- **identifier** (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
object

get_illuminator_geometry(*identifier=None*)

The illuminator shape defines the shape of the optical fibres, so it describes whether the illuminator is a point illuminator, or has a more continuous form. Illuminators can only have planar emitting surfaces.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

get_illuminator_geometry_type(*identifier=None*)

The illuminator geometry type defines the shape of the optical fibre (bundle) output. It determines the interpretation of the data in the illuminator geometry field. The following geometry types are currently supported:

```
- "CIRCULAR" - defined by a single value that determines the radius of the␣
↪circle
- "SPHERE" - defined by a single value that determines the radius of the sphere
- "CUBOID" - defined by three values that determine the extent of the cuboid in␣
↪x, y,nand z dimensions before the position and orientation transforms.
- "MESH" - defined by a STL-formatted string that determines the positions of␣
↪points and faces before the position and orientation transforms.
```

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_illuminator_ids()` → list

Returns a list of all IDs of the illumination elements that are added in this PAData instance.

Returns
a list of all ids of the illumination elements

Return type
list

`get_illuminator_orientation(identifier=None)`

The illuminator orientation defines the rotation of the illuminator in 3D cartesian coordinates [r1, r2, r3]. It is the normal of the planar illuminator surface.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_illuminator_position(identifier=None)`

The illuminator position defines the position of the illuminator centroid in 3D cartesian coordinates [x1, x2, x3] .

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_measurement_spatial_poses()`

Coordinates describing the position and orientation changes of the acquisition system relative to the measurement of reference (first measurement).

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_measurement_time_stamps()`

An array specifying the time at which a measurement was recorded.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_measurements_per_image()`

A single value describing the number of measurements that constitute the dataset corresponding to one image.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
int

`get_number_of_detectors()`

The number of detectors quantifies the number of transducer elements that are used in the respective PA imaging device. Each of these transducer elements is described by a set of detection geometry parameters.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
int

`get_number_of_illuminators()`

The number of illuminators quantifies the number of illuminators that are used in the respective PA imaging device. Each of these illuminators is described by a set of illumination geometry parameters.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
int

`get_overall_gain()`

A single value describing a factor used to modify the amplitude of the raw time series data.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
float

`get_photoacoustic_imaging_device_reference()`

A string referencing the UUID of the PA imaging device description as defined in the Device Metadata.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_pulse_energy()`

A value specifying the pulse energy used to generate the photoacoustic signal. If the pulse energies are averaged over many pulses, the average value must be specified.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_pulse_width(`*identifier=None*`)`

The pulse duration or pulse width describes the total length of a light pulse, measured as the time interval between the half-power points on the leading and trailing edges of the pulse.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_regions_of_interest()`

A list of named regions within the underlying 3D Cartesian coordinate system (cf. Device Metadata). Strings containing the region names are mapped to arrays that define either an approximate cuboid area (cf. Field of View) or a list of coordinates describing a set of 3D Cartesian coordinates surrounding the named region.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_sampling_rate()`

A single value referring to the rate at which samples of the analogue signal are taken to be converted into digital form.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
float

`get_scanning_method()`

A string representing the scanning method that is used. The following descriptions can be used: ("composite_scan", "full_scan"). This flag determines the way the metadatum "measurement" is defined.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
str

`get_sizes()`

The sizes field quantifies the number of data points in each of the dimensions specified in the dimensionality field. e.g. [128, 2560, 26] with a "2D+t" dimensionality.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_speed_of_sound()`

Either a single value representing the mean global speed of sound in the entire imaged medium or a 3D array representing a heterogeneous speed of sound map in the device coordinate system. This definition covers both the imaged medium and the coupling agent.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_temperature()`

An array describing the temperature of the imaged space (covering both the imaged medium and the coupling agent) for each measurement.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_time_gain_compensation()`

An array containing relative factors that have been used to correct the time series data for the effect of acoustic attenuation.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

`get_wavelength_range(identifier=None)`

The wavelength range quantifies the wavelength range that the illuminator is capable of generating by reporting three values: the minimum wavelength max, the maximum wavelength max and a metric for the accuracy accuracy: (min, max, accuracy). This parameter could for instance be (700, 900, 1.2), meaning that this illuminator can be tuned from 700 nm to 900 nm with an accuracy of 1.2 nm.

Parameters
identifier (*str*) – The ID of a specific illumination element. If *None* then all illumination elements are queried.

Returns
return value can be None, of the key was not found in the metadata dictionary.

Return type
np.ndarray

5.3 iohandler

`load_data(file_path: str)`

Loads a PAData instance from an IPASC-formatted HDF5 file.

Parameters
file_path (*str*) – Path of the HDF5 file to load the PAData from.

Returns
PAData instance containing all data and metadata read from the HDF5 file.

Return type
PAData

`write_data(file_path: str, pa_data: PAData, file_compression: Optional[str] = None)`

Saves a PAData instance into an HDF5 file according to the IPASC consensus format.

Parameters

- **file_path** (*str*) – Path of the file to save the dictionary in.

- **pa_data** (PAData) – Instance of the PAData class containing all information
- **file_compression** (*str*) – possible file compression for the hdf5 output file. Possible values are: gzip, lzf and szip.

Returns
This method does not return anything

Return type
None

5.4 qualitycontrol

The purpose of the qualitycontrol package is to provide the means for users to check their IPASC data for completeness and consistency.

It can also be used to check the data for general integrity.

class CompletenessChecker(*verbose: bool = False*, *log_file_path: Optional[str] = None*)

Bases: object

Tests a given AcquisitionMetadata dictionary or a given DeviceMetadata dictionary for completeness.

For these purposes, the check_acquisition_meta_data and check_device_meta_data methods can be used:

```
pa_data = #TODO load pa_data here
cc = CompletenessChecker(verbose=True)
acquisition_metadata_complete = cc.check_acquisition_meta_data(pa_data.meta_data_
↪acquisition)
device_metadata_complete = cc.check_device_meta_data(pa_data.meta_data_device)
```

Parameters

- **verbose** (*bool*) – A flag to indicate whether the log should be printed to the console.
- **log_file_path** (*str*) – A string with the path to where the log file should be written to. If 'None', then no log file is written.

check_acquisition_meta_data(*meta_data_dictionary: dict*) → bool

This function will evaluate the completeness of the given acquisition metadata. It can be used to generate a report to the console by setting *verbose* to True. When setting a file path to log_file, it will also save the report as a txt file in the designated path.

Parameters
meta_data_dictionary (*dict*) – A dictionary containing all acquisition meta data.

Raises

- **ValueError:** – if meta_data_dictionary was None
- **TypeError:** – if one of the arguments ws not of the correct type

Returns
True, if the meta_data_dictionary is complete

Return type
bool

check_device_meta_data(*device_meta_data: dict*)

This function will evaluate the completeness of the given device metadata. It can be used to generate a report to the console by setting *verbose* to True. When setting a file path to log_file, it will also save the report as a txt file in the designated path.

Parameters
device_meta_data (*dict*) – A dictionary containing all device meta data.

Raises

- **ValueError:** – if meta_data_dictionary was None
- **TypeError:** – if one of the arguments ws not of the correct type

Returns
True, if the meta_data_dictionary is complete

Return type
bool

static check_metadatum_from_dict(*dictionary: dict*, *metadatum:* MetaDatum)

Internal method to systematically test a metadata field.

Parameters

- **dictionary** (*dict*) – The dictionary supposedly containing the metadatum at the top level
- **metadatum** (MetaDatum) – The metadatum to test

Returns
A tuple with the log string and an integer that is 0 if everything was fine and 1 if there was an error.

Return type
(str, int)

class ConsistencyChecker(*verbose: bool = False*, *log_file_path: Optional[str] = None*)

Bases: object

The purpose of this class is to go beyond the capabilities of the CompletenessChecker and to test the consistency of the metadata. To this end, every meta datum is assigned a possible value range by definition. The Consistency checker unit_tests if the assigned values fall inside this value range.

Parameters

- **verbose** (*bool*) – A flag to indicate whether the log should be printed to the console.
- **log_file_path** (*str*) – A string with the path to where the log file should be written to. If 'None', then no log file is written.

check_acquisition_meta_data(*acquisition_meta_data: dict*) → bool

Tests the given dictionary with acquisition metadata for consistency.

Parameters
acquisition_meta_data (*dict*) – A dictionary containing all the acquisition metadata to check

Returns
Returns *True* if all data is consistent

Return type
bool

check_binary_data(*binary_data*) → bool

This method unit_tests if the given binary data has the correct data type and if each value in the binary data in a Number.

Parameters
binary_data (*np.ndarray*) – The binary data to check

check_device_meta_data(*device_meta_data: dict*) → bool

Tests the given dictionary with device metadata for consistency.

Parameters
device_meta_data (*dict*) – A dictionary containing all the device metadata to check

Returns
Returns *True* if all data is consistent

Return type
bool

quality_check_pa_data(*pa_data:* PAData, *verbose: bool = False*, *log_file_path: Optional[str] = None*) → bool

This is a convenience method that instantiates both a completeness and a consistency checker and evaluates the given PAData instance.

Parameters

- **pa_data** (PAData) – The PAData instance to check
- **verbose** (*bool*) – Specifies if the log report should be printed to the console
- **log_file_path** (*str*) – A path to a log file to write to

Returns
True if and only if all completeness and quality checks are passed.

Return type
bool

visualize_device(*device_dictionary: dict*, *save_path: Optional[str] = None*, *title: Optional[str] = None*, *only_show_xz: bool = False*, *show_legend: bool = True*)

Visualises a given device from the device_dictionary.

Parameters

- **device_dictionary** (*dict*) – The dictionary containing the device description.
- **save_path** (*str*) – Optional save_path with the path and file name to save the visualisation to.
- **title** (*str*) – Optional custom title for the plot.
- **only_show_xz** (*bool*) – Optional bool parameter specifying if only the first window should be shown instead of all
- **show_legend** (*bool*) – Optional parameter whether the figure legend should be shown (default: True)

CHAPTER

SIX

POSSIBLE METADATA ATTRIBUTES

Each metadatum is characterised by a series of attributes to describe and define its use and boundary conditions.
If necessary, further specifications by nested attributes can be given. All units of the metadata are stated in the {International System of Units} (SI units) unless otherwise specified.

6.1 Condition

Constraints of an attribute that limit its value range (e.g. the acquisition wavelengths must the of the same size as the acquired measurements)

6.2 Description

A short description of the attribute.

6.3 dtype

Data type of the attribute.

6.4 Measurement Device Attribute

A specific type of nested attribute that describes measurement device details if required. Measurement device attributes are always optional. They include:

6.4.1 Calibration Date

A timestamp referring to the date when the measurement device was last calibrated. Timestamps are given in seconds with the elapsed time since epoch (Jan 1st 1970, 00:00).

6.4.2 Measurement Device Manufacturer

A string literal describing the manufacturer of the measurement device, e.g. 'Thorlabs'.

6.4.3 Measurement Device Serial Number

A string literal comprising the serial number of the measurement device.

6.4.4 Measurement Device Type

A string literal describing the measurement device for this attribute, e.g. 'pyroelectric sensor' or 'wavemeter'.

6.5 Method Name

The name of the function/method that can be called in any programming language to obtain information on a specific attribute.

6.6 Necessity

'Minimal' or 'Report if present' condition for the metadatum. Minimal parameters are all parameters that are required to reconstruct an image from the raw time series data. Any additional information should be reported in the metadata if available.

6.7 Nested Attribute

A sub-attribute that further describes an attribute.

6.8 Units

SI units of the attribute if applicable.

CHAPTER

SEVEN

BINARY DATA METADATA

The binary data are formatted as: [detectors, samples, wavelengths, measurements]. Depending on the binary data metadata, the size of these arrays varies. The interpretation of the measurement field depends on the dimensionality field.

7.1 Data Type

The Data Type field represents the datatype of the binary data. This field is given in the C++ data type naming convention, e.g. 'short', 'unsigned short', 'int', 'unsigned int', 'long', 'unsigned long', 'long long', 'float', 'double', 'long double'.

Necessity	Minimal
dtype	String
Method Name	get_data_type()

7.2 Dimensionality

The Dimensionality field represents the definition of the 'measurement' field and can be either ['time', 'space', or 'time and space'].

Necessity	Minimal
dtype	String
Method Name	get_dimensionality()

7.3 Sizes

The Sizes field quantifies the number of data points in each of the dimensions specified in the dimensionality field. As such, it defines the respective sizes of each element of the binary data which are: [detectors, samples, wavelengths, measurements].

Necessity	Minimal
dtype	Integer array
Units	Dimensionless Quantity (the units can be inferred in combination with Dimensionality and the detection and illumination geometry).
Method Name	get_sizes()

CHAPTER

EIGHT

FILE CONTAINER FORMAT

The container format metadata refer to the inherent features of the file format which specify the organisation ofhow the different elements of metadata are combined in a computer file.

8.1 Encoding

The Encoding field defines the character set that was used to encode the binary data and the metadata, e.g. one of 'UTF-8', 'ASCII', 'CP-1252' etc.

Necessity	Minimal
dtype	String
Method Name	get_encoding()

8.2 Compression

The Compression field defines the compression method that was used to compress the binary data, e.g. one of 'raw', 'gzip etc.

Necessity	Minimal
dtype	String
Method Name	get_compression()

8.3 Universally Unique Identifier

The Universally Unique Identifier (UUID) field is a unique identifier of the data that can be referenced.

Necessity	Minimal
dtype	String
Condition	128-bit Integer displayed as a hexadecimal string in 5 groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters. The UUID is randomly generated using the UUID version 4 standard.
Method Name	get_data_UUID()

CHAPTER

NINE

ACQUISITION METADATA

9.1 A/D (Analog/Digital) Sampling Rate

The A/D Sampling Rate field refers to the rate at which samples of the analogue signal are taken to be converted into digital form.

Necessity	Minimal
dtype	Double
Units	Hertz [Hz] (samples / second)
Method Name	get_sampling_rate()

9.2 Acoustic Coupling Agent

The Acoustic Coupling Agent field is a string representation of the acoustic coupling agent that is used, e.g. D2O, H2O, gel, etc.

Necessity	Report if present
dtype	String
Method Name	get_coupling_agent()

9.3 Acquisition Optical Wavelengths

The Acquisition Optical Wavelengths field is an array of all wavelengths used for image acquisition.

Necessity	Minimal
dtype	Array
Units	Meters [m]
Method Name	get_wavelengths()

9.4 Element-dependent Gain

The Element-dependent Gain field is a 2D array that contains the relative factors which are used for apodization or detection element-wise sensitivity corrections.

Necessity	Report if present
dtype	Double array [num_detectors]
Units	Dimensionless unit
Condition	The element-dependent gain is a double array that has the same dimension as the number of detectors.
Method Name	get_element_dependent_gain()

9.5 Frequency Domain Filter

The Frequency Domain Filter field specifies an array defining the frequency threshold levels that are applied to filter the raw time series data, containing [lower, higher] -3 dB points of the filter in Hertz. [lower, -1] denotes a high-pass filter and [-1, higher] denotes a low-pass filter.

Necessity	Report if present
dtype	Double array
Units	Hertz [Hz] (samples / second)
Method Name	get_frequency_filter()

9.6 Measurements Per Image

The Measurements Per Image field specifies a single value describing the number of measurements that constitute the dataset corresponding to one image

Necessity	Report if present
dtype	Integer
Units	Dimensionless unit
Method Name	get_measurements_per_image()

9.7 Measurement Spatial Pose

The Measurement Spatial Pose field specifies coordinates describing the position and orientation changes of the acquisition system relative to the measurement of reference (first measurement). The entire coordinate system is moved based on the spatial positions. If the frame stays constant, N equals 0.

Necessity	Report if present
dtype	2D double array of 6D coordinates (N, 6)
Units	Meters [m]
Condition	Array size must be the same as the size of 'measurements' specified in the sizes field.
Method Name	get_measurement_spatial_pose()

9.8 Measurement Timestamps

The Measurement Timestamps field specifies the time at which a measurement was recorded.

Necessity	Report if present
dtype	Double array
Units	Seconds [s]
Condition	Array size must be the same as the size of 'measurements' specified in the sizes field. Timestamps are given in seconds with the elapsed time since epoch (Jan 1st 1970, 00:00).
Method Name	get_time_stamps()

9.9 Overall Gain

The Overall Gain field is a single value describing a factor used to modify the amplitude of the raw time series data.

Necessity	Report if present
dtype	Double
Units	Dimensionless unit
Method Name	get_overall_gain()

9.10 Photoacoustic Imaging Device Reference

The Photoacoustic Imaging Device Reference field specifies a reference to the UUID of the PA imaging device description as defined in part 1. This field will be used for future versions of the data format, where the device metadata may not be stored within the file but will be accessible via a web service.

Necessity	Report if present
dtype	String
Method Name	get_device_reference()

9.11 Pulse Laser Energy

The Pulse Laser Energy field specifies the pulse energy used to generate the PA signal. If the pulse energies are averaged over many pulses, the average value must be specified. If the pulse laser energy has already been accounted for, the array must read [0].

Necessity	Report if present
dtype	Double array
Units	Joule [J]
Condition	Array size must be the same as the size of 'measurements" specified in the sizes field, except for the case of [0]. It can also be of shape [detection_elements, measurements] in case that laser pulses are fired individually for each detection element.
Method Name	get_pulse_laser_energy()

9.12 Regions of Interest

The Regions of Interest field specifies a list of named regions within the underlying 3D Cartesian coordinate system (cf. Device Metadata). Strings containing the region names are mapped to arrays that define either an approximate cuboid area (cf. Field of View) or a list of coordinates describing a set of 3D Cartesian coordinates surrounding the named region. This field aims to facilitate the delineation of, e.g. distinct tissue types, potential lesions or phantom components. Regions of Interest are defined independently from the Field of View, and could be also outside the Field of View.

Necessity	Report if present
dtype	Dictionary [String, 2D double array (6, 3)] where the first number in the array represents the number of coordinates and the second number represents the coordinate values.
Units	Meter [m]
Method Name	get_region_of_interest()

9.13 Scanning Method

The Scanning Method field is a string representation of the scanning method that was used. The following descriptions can be used: ("composite_scan", "full_scan"). This flag determines the way the metadatum "measurement" is defined.

Necessity	Report if present
dtype	String
Method Name	get_scanning_method()

9.14 Speed of Sound

The Speed of Sound field specifies either a single value representing the mean global speed of sound in the entire imaged medium or a 3D array representing a heterogeneous speed of sound map in the device coordinate system. This definition covers both the imaged medium and the coupling agent.

Necessity	Report if present
dtype	Double or double array
Units	Meters per second [m/s]
Method Name	get_speed_of_sound()

9.15 Temperature Control

The Temperature Control field specifies the temperature of the imaged space (covering both the imaged medium and the coupling agent) for each measurement.

Necessity	Report if present
dtype	Double array
Units	Kelvin [K]
Condition	The temperature control array either has the same dimension as the number of 'measurements', or is a single value indicating a constant temperature over all measurements.
Method Name	get_temperature()

9.16 Time Gain Compensation

The Time Gain Compensation field is a 1D array that contains relative factors which are used to correct the time series data for the effect of acoustic attenuation.

Necessity	Report if present
dtype	Double array
Units	Dimensionless unit
Condition	The time gain compensation array has the same dimension as the samples dimension [samples]. It can also be of shape [detection_elements, samples] if measurements are acquired individually for each detection element.
Method Name	get_time_gain_compensation()

CHAPTER

TEN

DEVICE METADATA

The device metadata is split into three categories: Some general metadata parameters, metadata information on the detection geometry, and metadata information on the illumination geometry.

10.1 General Parameters

10.1.1 Field of View

The Field of View field defines an approximate cuboid (3D) area detectable by the PA imaging device in 3D cartesian coordinates [x1start, x1end, x2start, x2end, x3start, x3end]. A 2D Field of View can be defined by setting the start and end coordinate of the respective dimension to the same value.

Necessity	Minimal
dtype	2D double array of length 6
Units	Meter [m]
Method Name	get_field_of_view()

10.1.2 Number of Detection Elements

The Number of Detection Elements field quantifies the number of transducer elements used for detection in the PA imaging device. Each of these transducer elements is described by a set of detection geometry parameters.

Necessity	Minimal
dtype	Integer
Units	Dimensionless unit
Method Name	get_number_of_detection_elements()

10.1.3 Number of Illumination Elements

The Number of Illumination Elements field quantifies the number of illuminators that are used in the PA imaging device. Each of these illuminators is described by a set of illumination geometry parameters.

Necessity	Report if present
dtype	Integer
Units	Dimensionless unit
Method Name	get_number_of_illumination_elements()

10.1.4 Universally Unique Identifier

The Universally Unique Identifier (UUID) for the device that can be referenced.

Necessity	Minimal
dtype	String
Condition	128-bit Integer displayed as a hexadecimal string in 5 groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters. The UUID is randomly generated using the UUID version 4 standard.
Method Name	get_device_uuid()

10.2 Detection Element

10.2.1 Detector Geometry

The Detector Geometry field defines the shape of the detector elements. The data type and the contents of the shape field are determined by the Detector Geometry Type field. The given coordinates are interpreted relative to the Detector Position.

Necessity	Report if present
dtype	Double, double array, or byte array
Units	Meter [m]
Method Name	get_detector_geometry()

10.2.2 Detector Geometry Type

The Detector Geometry Type field defines the interpretation of the data in the detector geometry field. The following geometry types are currently supported:

- "CIRCULAR" - defined by a single value that determines the radius of the circle
- "SPHERE" - defined by a single value that determines the radius of the sphere
- "CUBOID" - defined by three values that determine the extent of the cuboid in x1, x2, and x3 dimensions before the position and orientation transforms.
- "MESH" - defined by an STL-formatted string that determines the positions of points and faces before the position and orientation transforms.

Necessity	Report if present
dtype	String
Method Name	get_detector_geometry_type()

10.2.3 Detector Orientation

The Detector Orientation field defines the direction unit vector of the detector in 3D Cartesian coordinates [xd, yd, zd] .

Necessity	Report if present
dtype	Double array
Units	Meter [m]
Method Name	get_detector_orientation()

10.2.4 Detector Position

The DetectorPosition field defines the position of the detection element centroid in 3D Cartesian coordinates [x1, x2, x3] .

Necessity	Minimal
dtype	Double array
Units	Meter [m]
Method Name	get_detector_position()

10.2.5 Detection Element Properties

Angular Response

The Angular Response field characterises the angular sensitivity of the detection element to the incident angle (relative to the element's orientation) of the incoming pressure wave. If only one value (the angle [a]) is given, the value can be interpreted as a=limiting angle (where the response drops to -6 dB).

Necessity	Report if present
dtype	Array with two components, where the first component is the incident angle in radians and the second component is the normalised response value.
Units	Radians [rad], Normalised Units (to the maximum efficiency)
Method Name	get_angular_response()

Frequency Response

The Frequency Response field describes a function of frequency that characterises the response of the detection element with respect to the frequency of incident pressure waves. If the response is only sparsely defined, the values can be linearly interpolated between the closest neighbours. If the value is of shape [c, b], it can be interpreted as c=centre frequency and b=bandwidth (measured at -6 dB).

Necessity	Report if present
dtype	Array with two components, where the first component is the frequency (in Hertz [s-1]) and the second component is the response value (in normalised units).
Units	Hertz [s-1], normalised units (to the maximum intensity)
Method Name	get_frequency_response()

10.3 Illumination Element

10.3.1 Illuminator Geometry

The Illuminator Geometry field defines the numerical geometry of the optical fibre (bundle) output. The data type and content of this metadatum are determined by the illuminator geometry type field. The given coordinates are interpreted relative to the illuminator position.

Necessity	Report if present
dtype	Double, double array, or byte array
Units	Meter [m]
Method Name	get_illuminator_geometry()

10.3.2 Illuminator Geometry Type

The Illuminator Geometry Type field defines the shape of the optical fibre (bundle) output. It determines the interpretation of the data in the illuminator geometry field. The following geometry types are currently supported:

- "CIRCULAR" - defined by a single value that determines the radius of the circle.
- "SPHERE" - defined by a single value that determines the radius of the sphere.
- "CUBOID" - defined by three values that determine the extent of the cuboid in x1, x2, and x3 dimensions before the position and orientation transforms.
- "MESH" - defined by an STL-formatted string that determines the positions of points and faces before the position and orientation transforms.

Necessity	Report if present
dtype	String
Method Name	get_illuminator_geometry_type()

10.3.3 Illuminator Orientation

The Illuminator Orientation field defines the direction unit vector of the illuminator in 3D Cartesian coordinates [x1d, x2d, x3d] . This unit vector is the normal of the planar illuminator surface.

Necessity	Report if present
dtype	1D double array
Units	Meter [m]
Method Name	get_illuminator_orientation()

10.3.4 Illuminator Position

The Illuminator Position field defines the position of the illuminator centroid in 3D cartesian coordinates [x1, x2, x3] .

Necessity	Report if present
dtype	1D double array
Units	Meter [m]
Method Name	get_illuminator_position()

10.3.5 Illuminator Properties

Beam Divergence Angles

The Beam Divergence Angles field represents the opening angles of the laser beam from the illuminator shape with respect to the orientation vector. This angle is represented by the standard deviation of the beam divergence.

Necessity	Report if present
dtype	Double
Units	Radians [rad]
Method Name	get_beam_divergence()

Beam Intensity Profile

The Beam Intensity Profile field is a function of a spatial position that specifies the relative laser beam intensity according to the planar emitting surface of the illuminator shape at the distance defined in intensity profile distance. For points between specified positions, it is assumed that the values are linearly interpolated from their closest neighbours. The positions are generally in 2D.

Necessity	Report if present
dtype	Array of two double arrays [positions, intensities] with intensities and their corresponding positions.
Units	Normalised units (to the maximum intensity)
Method Name	get_beam_profile()

Intensity Profile Distance

The Intensity Profile Distance field describes the distance from the light source for measuring its beam intensity profile. This distance is to be measured from the Illuminator Position along with the Illuminator Orientation.

Necessity	Report if present
dtype	Double
Units	Meters [m]
Method Name	get_beam_profile_distance()

Laser Energy Profile

The Laser Energy Profile field is a discretized function of the wavelength (nm) describing the laser energy of the illuminator. Thereby, systematic differences in multispectral image acquisitions can be accounted for.

Necessity	Report if present
dtype	Array of two 1D double arrays [wavelengths, energies], where the first array comprises the wavelengths and the second array comprises the laser energies.
Units	Joule [J]
Condition	The laser energy profile function is well defined and non-negative in the wavelength range.
Method Name	get_energy_profile()

Laser Stability Profile

The Laser Stability Profile field is a function of the wavelength (nm) and represents the standard deviation of the pulse-to-pulse laser energy of the illuminator.

Necessity	Report if present
dtype	Array of two 1D double arrays [wavelengths, energies], where the first array comprises the wavelengths and the second array comprises the laser energies.
Units	Joule [J]
Condition	The laser stability profile function is well defined and non-negative in the wavelength range.
Method Name	get_stability_profile()

Pulse Duration / Width

The Pulse Duration /Width field describes the total length of a laser pulse measured as the time interval between the half-power points on the leading and trailing edges of the pulse.

Necessity	Report if present
dtype	Double
Units	Seconds [s]
Method Name	get_pulse_width()

Wavelength Range

The Wavelength Range field quantifies the wavelengths that can be generated by the illuminator. Three values can be reported: the minimum wavelength max, the maximum wavelength max and a metric for the accuracy accuracy: (min, max, accuracy). These parameters could be for instance (700, 900, 1.2), meaning that this illuminator can be tuned from 700 nm to 900 nm with an accuracy of 1.2 nm. Single-wavelength elements are specified as: (actual, actual, accuracy).

Necessity	Report if present
dtype	1D double array
Units	Meters [m]
Method Name	get_wavelength_range()

PYTHON MODULE INDEX

p

INDEX

F

G

H

I

Q

R

S

T

U

V

W