PythonSOFA

Release v0.1.2-7-g739d491

Jannika Lossner

2020-03-03

Contents

1 Examples **1**
- 1.1 Open and plot HRTF . . . 2
- 1.2 Create new SimpleFreeFieldHRIR .sofa file . . . 6

2 API Documentation **8**
- 2.1 sofa.access . . . 9
- 2.2 sofa.conventions . . . 13
- 2.3 sofa.datatypes . . . 13
- 2.4 sofa.roomtypes . . . 17
- 2.5 sofa.spatial . . . 20

3 Version History **24**

A Python API for reading, writing and creating SOFA files as defined by the SOFA conventions (version 1.0) found at https://www.sofaconventions.org/.

Documentation: https://python-sofa.readthedocs.io/

Source code and issue tracker: https://github.com/spatialaudio/python-sofa/

License: MIT – see the file `LICENSE` for details.

Quick start:

- Install Python 3
- `python3 -m pip install python-sofa --user`
- Check out the examples in the documentation

1 Examples

The following section was generated from `doc/examples/SOFA-file-access.ipynb`

[1]:
```
import sys
sys.path.insert(0, '../../src')
import sofa
print(sofa)
```

```
<module 'sofa' from '../../src/sofa/__init__.py'>
```

[2]:
```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import numpy as np
%matplotlib inline
```

[3]:
```
def plot_coordinates(coords, title):
    x0 = coords
    n0 = coords
    fig = plt.figure(figsize=(15, 15))
    ax = fig.add_subplot(111, projection='3d')
    q = ax.quiver(x0[:, 0], x0[:, 1], x0[:, 2], n0[:, 0],
                  n0[:, 1], n0[:, 2], length=0.1)
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
    plt.title(title)
    return q
```

1.1 Open and plot HRTF

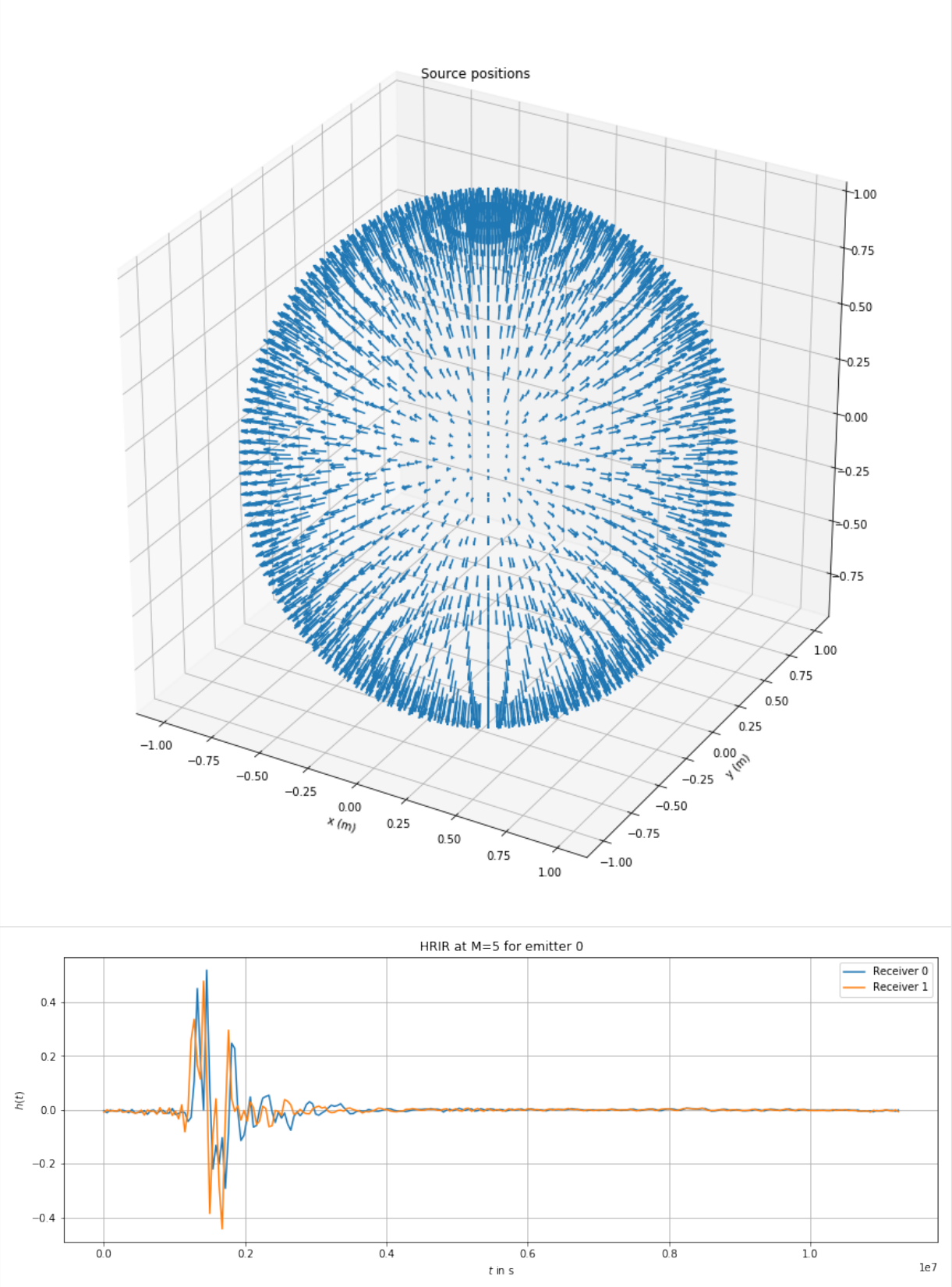
[4]:
```
HRTF_path = "MRT01.sofa"
HRTF = sofa.Database.open(HRTF_path)
HRTF.Metadata.dump()

# plot Source positions
source_positions = HRTF.Source.Position.get_values(system="cartesian")
plot_coordinates(source_positions, 'Source positions');

# plot Data.IR at M=5 for E=0
measurement = 5
emitter = 0
legend = []

t = np.arange(0,HRTF.Dimensions.N)*HRTF.Data.SamplingRate.get_values(indices={"M":
 ↪measurement})

plt.figure(figsize=(15, 5))
for receiver in np.arange(HRTF.Dimensions.R):
    plt.plot(t, HRTF.Data.IR.get_values(indices={"M":measurement, "R":receiver, "E":
 ↪emitter}))
    legend.append('Receiver {0}'.format(receiver))
plt.title('HRIR at M={0} for emitter {1}'.format(measurement, emitter))
plt.legend(legend)
plt.xlabel('$t$ in s')
plt.ylabel(r'$h(t)$')
plt.grid()

HRTF.close()
```

```
APIName: ARI SOFA API for Matlab/Octave
APIVersion: 1.0.2
```

(continues on next page)

(continued from previous page)

```
ApplicationName: ITA-Toolbox
ApplicationVersion: 7
AuthorContact: rbo (@akustik.rwth-aachen.de)
Comment:
Conventions: SOFA
DataType: FIR
DatabaseName: ITA HRTF-database
DateCreated: 20-Oct-2016
DateModified: 2017-08-10 16:13:37
History: R. Bomhardt, M. de la Fuente Klein, and J. Fels: A high-resolution head-related␣
 ↪transfer function and three-dimensional ear model database, Proceedings of Meetings on␣
 ↪Acoustics 29, 050002 (2016)
License: Creative Commons Attribution-NonCommercial-ShareAlike 4.0 (CC BY-NC-SA 4.0)
ListenerShortName: MRT01
Organization: Institute of Technical Acoustics, RWTH Aachen University
Origin:
References: R. Bomhardt, M. de la Fuente Klein, and J. Fels: A high-resolution head-
 ↪related transfer function and three-dimensional ear model database, Proceedings of␣
 ↪Meetings on Acoustics 29, 050002 (2016)
RoomDescription: 11m x 5.9m x 4.5m
RoomType: free field
SOFAConventions: SimpleFreeFieldHRIR
SOFAConventionsVersion: 1.0
Title: ITA HRTF
Version: 1.0
```



Retrieving coordinates

Coordinates may be retrieved using one of three methods:

- `SpatialObject.get_values(...)` returns the local coordinates as stored in the file
- `SpatialObject.get_global_values(...)` returns the global coordinates by removing the position and rotation of the reference object (reference object for Emitters is Source, reference object for Receivers is Listener)
- `SpatialObject.get_relative_values(SpatialObject reference,...)` returns the coordinates in the local coordinate system of the specified reference object by applying position and rotation of the reference to the global coordinates of the object, `reference=None` is equivalent to `SpatialObject.get_global_values(...)`

[5]:
```
HRTF_path = "MRT01.sofa"
HRTF = sofa.Database.open(HRTF_path)

print("Source positions 0 to 4 in spherical coordinates")
print(np.round(HRTF.Source.Position.get_values(indices={"M":slice(5)}, system="spherical",␣
↪angle_unit="degree"),2))

print("Stationary local Emitter position")
print(np.round(HRTF.Emitter.Position.get_values(system="spherical", angle_unit="degree"),
↪2))

print("Global Emitter positions 0 to 4")
print(np.round(HRTF.Emitter.Position.get_global_values(indices={"M":slice(5)}, system=
↪"spherical", angle_unit="degree"),2))

print("Stationary Listener position")
print(np.round(HRTF.Listener.Position.get_values(system="spherical", angle_unit="degree"),
↪2))

print("Listener positions 0 to 4 relative to Emitter")
print(np.round(HRTF.Listener.Position.get_relative_values(HRTF.Emitter, indices={"M":
↪slice(5)}, system="spherical", angle_unit="degree"),2))

HRTF.close()
```

```
Source positions 0 to 4 in spherical coordinates
[[ -0.   -66.24   1.  ]
 [ -0.   -61.2    1.  ]
 [ -0.   -56.16   1.  ]
 [ -0.   -51.12   1.  ]
 [ -0.   -46.08   1.  ]]
Stationary local Emitter position
[[[0.]
  [0.]
  [0.]]]
Global Emitter positions 0 to 4
[[[ -0.    -0.    -0.    -0.    -0.  ]
  [-66.24 -61.2  -56.16 -51.12 -46.08]
  [  1.     1.     1.     1.     1.  ]]]
Stationary Listener position
[[0. 0. 0.]]
Listener positions 0 to 4 relative to Emitter
[[[180.    66.24   1.  ]
  [180.    61.2    1.  ]
  [180.    56.16   1.  ]
  [180.    51.12   1.  ]
  [180.    46.08   1.  ]]]
```

1.2 Create new SimpleFreeFieldHRIR .sofa file

When creating a new .SOFA file, a convention must be provided by name. Providing dimensions at this point is possible, but not required.

[6]:
```
print(sofa.conventions.implemented())
```

```
['GeneralFIR', 'GeneralTF', 'SimpleFreeFieldHRIR', 'GeneralFIRE', 'MultiSpeakerBRIR',
↪'SimpleFreeFieldTF', 'SimpleFreeFieldSOS', 'SingleRoomDRIR']
```

[7]:
```
HRIR_path = "free_field_HRIR.sofa"
measurements = 5
data_length = 1000
max_string_length = 128

# we will add this one later
receivers = 2
```

[8]:
```
HRIR = sofa.Database.create(HRIR_path, "SimpleFreeFieldHRIR",
                            dimensions={"M": measurements, "N": data_length, "S": max_
↪string_length})
```

Next, all spatial objects (Listener, Receiver, Source, Emitter) must be initialized to set which of their coordinates will stay fixed over the measurements, and which will vary. This information is provided in the `fixed` and `variances` list of the initialization method. Defining the Position is required for all spatial objects, and conventions can extend the requirements.

[9]:
```
try: HRIR.Listener.initialize()
except Exception as e: print("Initializing without Position error: {0}".format(e))

try: HRIR.Listener.initialize(fixed=["Position"])
except Exception as e: print("Initializing with incomplete convention requirements: {0}".
↪format(e))

HRIR.Listener.initialize(fixed=["Position", "View", "Up"])
print("Successfully initialized Listener with fixed Position, View and Up.")
```

```
Initializing without Position error: Listener.initialize: Missing 'Position' in fixed or␣
↪variances argument
Initializing with incomplete convention requirements: must have Listener Up and View
Successfully initialized Listener with fixed Position, View and Up.
```

It is also possible to initialize variables later on that were not required at the start by calling the `initialize_coordinates` method again. Attempting to initialize a coordinate that already exists is ignored.

[10]:
```
HRIR.Source.initialize(variances=["Position"])
HRIR.Source.initialize_coordinates(variances=["View"])

HRIR.Source.initialize_coordinates(fixed=["Position"]) # no change, message or output.
```

If the number of emitters and receivers was not defined by the convention or when the file was created, it can manually be defined by including the `count` keyword argument in `initialize`.

[11]:
```
HRIR.Receiver.initialize(fixed=["Position"], count=receivers)

try: HRIR.Emitter.initialize(fixed=["Position"], count=2)
except Exception as e: print("Initializing with Emitter count not allowed in convention␣
↪error: {0}".format(e))
HRIR.Emitter.initialize(fixed=["Position"])
```

```
Initializing with Emitter count not allowed in convention error: must have 1 Emitter
```

Finally, we need to initialize the data type as well. This can only be done once the receiver count (and emitter count for FIRE) have been defined, so after the initialization of spatial objects or after a call to `create` that includes the relevant dimensions. Data that may be fixed or varying can be declared as varying in the `variances` list keyword argument.

The sample count N may be included at this point as the keyword argument `sample_count` if it is not already defined.

[12]:
```
print(HRIR.Data.Type)
HRIR.Data.initialize(variances=["Delay"])
```

```
FIR
```

The room data may be initialized at any point. As with any `ProxyObject`, additional SOFA attributes and variables may be created as well.

[13]:
```
HRIR.Room.Type = "shoebox"
HRIR.Room.initialize(variances=["CornerA", "CornerB"])

HRIR.Room.create_attribute("Location", "various recording locations")
HRIR.Room.create_variable("Temperature", ("M",))
HRIR.Room.Temperature.Units = "kelvin"
HRIR.Room.Temperature = 150
HRIR.Room.create_string_array("Description", ("M", "S"))

print(HRIR.Room.Location)
print(HRIR.Room.Temperature.get_values(), HRIR.Room.Temperature.Units)
```

```
various recording locations
[150. 150. 150. 150. 150.] kelvin
```

The file is now fully initialized and ready for use. You can take a look at a list of all attributes, variables and dimensions using the appropriate `.dump()` functions.

[14]:
```
print("Attributes and metadata")
HRIR.Metadata.dump()
```

```
Attributes and metadata
APIName: python-SOFA
APIVersion: 0.2
AuthorContact:
Conventions: SOFA
DataType: FIR
DatabaseName:
DateCreated: 2020-03-03 15:33:24
DateModified:
EmitterDescription:
License: No license provided, ask the author for permission
ListenerDescription:
ListenerShortName:
Organization:
ReceiverDescription:
RoomLocation: various recording locations
RoomType: shoebox
SOFAConventions: SimpleFreeFieldHRIR
SOFAConventionsVersion: 1.0
SourceDescription:
Title:
Version: 1.0
```

[15]:
```
print("Dimensions")
HRIR.Dimensions.dump()
```

```
Dimensions
M: 5
N: 1000
S: 128
I: 1
C: 3
R: 2
E: 1
```

[16]:
```
print("Variables")
HRIR.Variables.dump()
```

```
Variables
Data.Delay: ('M', 'R')
Data.IR: ('M', 'R', 'N')
Data.SamplingRate: ('I',)
EmitterPosition: ('E', 'C', 'I')
ListenerPosition: ('I', 'C')
ListenerUp: ('I', 'C')
ListenerView: ('I', 'C')
ReceiverPosition: ('R', 'C', 'I')
RoomCornerA: ('M', 'C')
RoomCornerB: ('M', 'C')
RoomDescription: ('M', 'S')
RoomTemperature: ('M',)
SourcePosition: ('M', 'C')
SourceView: ('M', 'C')
```

[17]:
```
HRIR.close()
```

doc/examples/SOFA-file-access.ipynb ends here.

2 API Documentation

Python SOFA API for reading, writing and creating .sofa files.

class sofa.**Database**

Read and write NETCDF4 files following the SOFA specifications and conventions

static create(*path*, *convention*, *dimensions=None*)

Create a new .sofa file following a SOFA convention

Parameters

- **path** (*str*) – Relative or absolute path to .sofa file
- **convention** (*str*) – Name of the SOFA convention to create, see *sofa.conventions.implemented()*
- **dimensions** (*dict or int, optional*) – Number of measurements or dict of dimensions to define (standard dimensions: "M": measurements, "R": receivers, "E": emitters, "N": data length)

Returns database (*sofa.Database*)

static open(*path*, *mode='r'*, *parallel=False*)

Parameters

- **path** (*str*) – Relative or absolute path to .sofa file
- **mode** (*str, optional*) – File access mode ('r': readonly, 'r+': read/write)
- **parallel** (*bool, optional*) – Whether to open the file with parallel access enabled (requires parallel-enabled netCDF4)

Returns **database** (*sofa.Database*)

close()

save()

convention

Data
DataType specific access for the measurement data, see *sofa.datatypes*

Dimensions
sofa.access.Dimensions for the database dimensions

Listener
sofa.spatial.SpatialObject for the Listener

Source
sofa.spatial.SpatialObject for the Source

Receiver
sofa.spatial.SpatialObject for the Receiver(s)

Emitter
sofa.spatial.SpatialObject for the Emitter(s)

Room
RoomType specific access for the room data, see *sofa.roomtypes*

Metadata
sofa.access.Metadata for the database metadata

Variables
sofa.access.DatasetVariables for direct access to database variables

2.1 sofa.access

class sofa.access.**DatasetVariables**(*database*)

get_variable(*name*)

Parameters **name** (*str*) – Name of the variable

Returns **value** (*sofa.access.Variable*) – Access object for the variable

get_string_array(*name*)

Parameters **name** (*str*) – Name of the string array

Returns **value** (*sofa.access.StringArray*) – Access object for the string array

create_variable(*name*, *dims*, *data_type='d'*, *fill_value=0*)

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.Variable*) – Access object for the variable

`create_string_array`(*name*, *dims*)

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.StringArray*) – Access object for the string array

`list_variables`()

Returns **attrs** (*list*) – List of the existing dataset variable and string array names

`dump`()
Prints all variables and their dimensions

`class` sofa.access.`Dimensions`(*dataset*)
Dimensions specified by SOFA as int

`C`
Coordinate dimension size

`I`
Scalar dimension size

`M`
Number of measurements

`R`
Number of receivers

`E`
Number of emitters

`N`
Number of data samples per measurement

`S`
Largest data string size

`get_dimension`(*dim*)

`create_dimension`(*dim*, *size*)

`list_dimensions`()

`dump`()
Prints all dimension sizes

`class` sofa.access.`Metadata`(*dataset*)

`get_attribute`(*name*)

Parameters **name** (*str*) – Name of the attribute

Returns **value** (*str*) – Value of the attribute

`set_attribute`(*name*, *value*)

Parameters

- **name** (*str*) – Name of the attribute
- **value** (*str*) – New value of the attribute

`create_attribute`(*name*, *value=''*)

Parameters

- **name** (*str*) – Name of the attribute
- **value** (*str, optional*) – New value of the attribute

`list_attributes`()

Returns attrs (*list*) – List of the existing dataset attribute names

`dump`()

Prints all metadata attributes

class sofa.access.**ProxyObject**(*database*, *name*)

Proxy object that provides access to variables and attributes of a name group in the netCDF4 dataset

database

name

dataset

`create_attribute`(*name*, *value=''*)

Creates the attribute in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **value** (*str, optional*) – Initial value of the attribute

`create_variable`(*name*, *dims*, *data_type='d'*, *fill_value=0*)

Creates the variable in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns value (*sofa.access.Variable*) – Access object for the variable

`create_string_array`(*name*, *dims*)

Creates the string array in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns value (*sofa.access.StringArray*) – Access object for the string array

class sofa.access.**StringArray**(*database*, *name*)

`initialize`(*dims*, *data_type='c'*, *fill_value='\xoo'*)

Create the zero-padded character array in the underlying netCDF4 dataset. Dimension 'S' must be the last dimension, and is appended if not included in dims.

`get_values`(*indices=None*, *dim_order=None*)

Parameters

- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be returned, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Desired order of dimensions in the output array

Returns **values** (*np.ndarray*) – Requested array range in regular or desired dimension order, if provided

`axis`(*dim*)

Parameters **dim** (*str*) – Name of the dimension

Returns **axis** (*int*) – Index of the dimension axis or None if unused

`database`

`dimensions`()

Returns **dimensions** (*tuple of str*) – Variable dimension names in order

`exists`()

Returns **exists** (*bool*) – True if variable exists, False otherwise

`name`

`set_values`(*values*, *indices=None*, *dim_order=None*, *repeat_dim=None*)

Parameters

- **values** (*np.ndarray*) – New values for the array range
- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be set, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Dimension names in provided order, regular order assumed
- **repeat_dim** (*tuple of str, optional*) – Tuple of dimension names along which to repeat the values

`class` sofa.access.`Variable`(*database*, *name*)

`Units`
Units of the values

`axis`(*dim*)

Parameters **dim** (*str*) – Name of the dimension

Returns **axis** (*int*) – Index of the dimension axis or None if unused

`database`

`dimensions`()

Returns **dimensions** (*tuple of str*) – Variable dimension names in order

`exists`()

Returns **exists** (*bool*) – True if variable exists, False otherwise

`get_values`(*indices=None*, *dim_order=None*)

Parameters

- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be returned, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Desired order of dimensions in the output array

Returns **values** (*np.ndarray*) – Requested array range in regular or desired dimension order, if provided

initialize(*dims*, *data_type='d'*, *fill_value=0*)
Create the variable in the underlying netCDF4 dataset

name

set_values(*values*, *indices=None*, *dim_order=None*, *repeat_dim=None*)

Parameters

- **values** (*np.ndarray*) – New values for the array range
- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be set, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Dimension names in provided order, regular order assumed
- **repeat_dim** (*tuple of str, optional*) – Tuple of dimension names along which to repeat the values

2.2 sofa.conventions

sofa.conventions.**implemented**()

Returns *list* – Names of implemented SOFA conventions

2.3 sofa.datatypes

Classes for accessing DataType-specific measurement data.

sofa.datatypes.**implemented**()

Returns *list* – Names of implemented SOFA data types

class sofa.datatypes.**FIR**(*database*)
Finite Impulse Response data type

IR [*sofa.access.Variable*] Discrete time impulse responses, dimensions ('M', 'R', 'N')

Delay [*sofa.access.Variable*] Broadband delay in units of dimension 'N', dimensions ('I', 'R') or ('M', 'R')

SamplingRate [*sofa.access.Variable*] Sampling rate, dimensions ('I') or ('M'), with attribute "Units"

Type
SOFA data type

create_attribute(*name*, *value=''*)
Creates the attribute in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **value** (*str, optional*) – Initial value of the attribute

create_string_array(*name*, *dims*)
Creates the string array in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.StringArray*) – Access object for the string array

`create_variable`(*name*, *dims*, *data_type='d'*, *fill_value=0*)
Creates the variable in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.Variable*) – Access object for the variable

`database`

`dataset`

`initialize`(*sample_count=None*, *variances=[]*, *string_length=None*)
Create the necessary variables and attributes

Parameters

- **sample_count** (*int, optional*) – Number of samples per measurement, mandatory if dimension N has not been defined
- **variances** (*list*) – Names of the variables that vary along dimension M
- **string_length** (*int, optional*) – Size of the longest data string

`name`

`optional_variance_names`()
Returns a list of standardized data elements that may vary between measurements

`class` sofa.datatypes.`FIRE`(*database*)
Finite Impulse Response per Emitter data type

IR [*sofa.access.Variable*] Discrete time impulse responses, dimensions ('M', 'R', 'E', 'N')

Delay [*sofa.access.Variable*] Broadband delay in units of dimension 'N', dimensions ('I', 'R', 'E') or ('M', 'R', 'E')

SamplingRate [*sofa.access.Variable*] Sampling rate, dimensions ('I') or ('M'), with attribute "Units"

`Type`
SOFA data type

`create_attribute`(*name*, *value=''*)
Creates the attribute in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **value** (*str, optional*) – Initial value of the attribute

`create_string_array`(*name*, *dims*)
Creates the string array in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.StringArray*) – Access object for the string array

`create_variable`(*name*, *dims*, *data_type='d'*, *fill_value=0*)
Creates the variable in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.Variable*) – Access object for the variable

`database`

`dataset`

`initialize`(*sample_count=None*, *variances=[]*, *string_length=None*)
Create the necessary variables and attributes

Parameters

- **sample_count** (*int, optional*) – Number of samples per measurement, mandatory if dimension N has not been defined
- **variances** (*list*) – Names of the variables that vary along dimension M
- **string_length** (*int, optional*) – Size of the longest data string

`name`

`optional_variance_names`()
Returns a list of standardized data elements that may vary between measurements

`class` sofa.datatypes.`SOS`(*database*)
Second Order Sections data type

SOS [*sofa.access.Variable*] Second order sections, dimensions ('M', 'R', 'N')

Delay [*sofa.access.Variable*] Broadband delay in samples resulting from SamplingRate, dimensions ('M', 'R')

SamplingRate [*sofa.access.Variable*] Sampling rate, dimensions ('I') or ('M'), with attribute "Units"

`initialize`(*sample_count=None*, *variances=[]*, *string_length=None*)
Create the necessary variables and attributes

Parameters

- **sample_count** (*int, optional*) – Number of samples per measurement, mandatory if dimension N has not been defined
- **variances** (*list*) – Names of the variables that vary along dimension M
- **string_length** (*int, optional*) – Size of the longest data string

`Type`
SOFA data type

`create_attribute`(*name*, *value=''*)
Creates the attribute in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **value** (*str, optional*) – Initial value of the attribute

`create_string_array`(*name*, *dims*)
Creates the string array in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable

- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.StringArray*) – Access object for the string array

`create_variable`(*name*, *dims*, *data_type='d'*, *fill_value=0*)
Creates the variable in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.Variable*) – Access object for the variable

`database`

`dataset`

`name`

`optional_variance_names`()
Returns a list of standardized data elements that may vary between measurements

`class` sofa.datatypes.`TF`(*database*)
Transfer Function data type

Real [*sofa.access.Variable*] Real part of the complex spectrum, dimensions ('M', 'R', 'N')

Imag [*sofa.access.Variable*] Imaginary part of the complex spectrum, dimensions ('M', 'R', 'N')

N [*sofa.access.Variable*] Frequency values, dimension ('N',), with attributes "LongName" and "Units"

`N`
Frequency values

`Type`
SOFA data type

`create_attribute`(*name*, *value=''*)
Creates the attribute in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **value** (*str, optional*) – Initial value of the attribute

`create_string_array`(*name*, *dims*)
Creates the string array in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.StringArray*) – Access object for the string array

`create_variable`(*name*, *dims*, *data_type='d'*, *fill_value=0*)
Creates the variable in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.Variable*) – Access object for the variable

database

dataset

initialize(*sample_count=None*, *variances=[]*, *string_length=None*)
Create the necessary variables and attributes

Parameters

- **sample_count** (*int, optional*) – Number of samples per measurement, mandatory if dimension N has not been defined
- **variances** (*list*) – Names of the variables that vary along dimension M
- **string_length** (*int, optional*) – Size of the longest data string

name

optional_variance_names()
Returns a list of standardized data elements that may vary between measurements

2.4 sofa.roomtypes

Classes for accessing RoomType-specific data.

sofa.roomtypes.**implemented**()

Returns *list* – Names of implemented SOFA room types

class sofa.roomtypes.**FreeField**(*database*)

Type
SOFA data type

create_attribute(*name*, *value=''*)
Creates the attribute in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **value** (*str, optional*) – Initial value of the attribute

create_string_array(*name*, *dims*)
Creates the string array in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.StringArray*) – Access object for the string array

create_variable(*name*, *dims*, *data_type='d'*, *fill_value=0*)
Creates the variable in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.Variable*) – Access object for the variable

database

dataset

`initialize`(*variances=[]*, *string_length=None*)
Create the necessary variables and attributes

Parameters

- **measurement_count** (*int*) – Number of measurements
- **sample_count** (*int*) – Number of samples per measurement
- **variances** (*list*) – Names of the variables that vary along dimension M
- **string_length** (*int, optional*) – Size of the longest data string

`name`

`optional_variance_names`()
Returns a list of standardized data elements that may vary between measurements

`class` sofa.roomtypes.`Reverberant`(*database*)

`initialize`(*variances=[]*, *string_length=None*)
Create the necessary variables and attributes

Parameters

- **measurement_count** (*int*) – Number of measurements
- **sample_count** (*int*) – Number of samples per measurement
- **variances** (*list*) – Names of the variables that vary along dimension M
- **string_length** (*int, optional*) – Size of the longest data string

`Type`
SOFA data type

`create_attribute`(*name*, *value=''*)
Creates the attribute in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **value** (*str, optional*) – Initial value of the attribute

`create_string_array`(*name*, *dims*)
Creates the string array in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns value (*sofa.access.StringArray*) – Access object for the string array

`create_variable`(*name*, *dims*, *data_type='d'*, *fill_value=0*)
Creates the variable in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns value (*sofa.access.Variable*) – Access object for the variable

`database`

`dataset`

`name`

`optional_variance_names`()
Returns a list of standardized data elements that may vary between measurements

`class` sofa.roomtypes.**Shoebox**(*database*)
Shoebox room type

CornerA [*sofa.spatial.Coordinates*] First corner of room cuboid, dimensions ('I', 'C') or ('M', 'C')

CornerB [*sofa.spatial.Coordinates*] Opposite corner of room cuboid, dimensions ('I', 'C') or ('M', 'C')

`CornerA`
First corner of room cuboid

`CornerB`
Opposite corner of room cuboid

`initialize`(*variances=[]*, *string_length=None*)
Create the necessary variables and attributes

Parameters

- **measurement_count** (*int*) – Number of measurements
- **sample_count** (*int*) – Number of samples per measurement
- **variances** (*list*) – Names of the variables that vary along dimension M
- **string_length** (*int, optional*) – Size of the longest data string

`Type`
SOFA data type

`create_attribute`(*name*, *value=''*)
Creates the attribute in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **value** (*str, optional*) – Initial value of the attribute

`create_string_array`(*name*, *dims*)
Creates the string array in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.StringArray*) – Access object for the string array

`create_variable`(*name*, *dims*, *data_type='d'*, *fill_value=0*)
Creates the variable in the netCDF4 dataset with its full name self.name+name

Parameters

- **name** (*str*) – Name of the variable
- **dims** (*tuple(str)*) – Dimensions of the variable

Returns **value** (*sofa.access.Variable*) – Access object for the variable

`database`

`dataset`

name

optional_variance_names()
Returns a list of standardized data elements that may vary between measurements

2.5 sofa.spatial

sofa.spatial.**sph2cart**(*alpha*, *beta*, *r*)
Spherical to cartesian coordinate transform.

$$x = r\cos\alpha\sin\beta$$
$$y = r\sin\alpha\sin\beta$$
$$z = r\cos\beta$$

with $\alpha \in [0, 2\pi), \beta \in [-\frac{\pi}{2}, \frac{\pi}{2}], r \geq 0$

Parameters

- **alpha** (*float or array_like*) – Azimuth angle in radians
- **beta** (*float or array_like*) – Elevation angle in radians (with o denoting azimuthal plane)
- **r** (*float or array_like*) – Radius

Returns

- **x** (float or numpy.ndarray[1]) – x-component of Cartesian coordinates
- **y** (float or numpy.ndarray[2]) – y-component of Cartesian coordinates
- **z** (float or numpy.ndarray[3]) – z-component of Cartesian coordinates

sofa.spatial.**cart2sph**(*x*, *y*, *z*)
Cartesian to spherical coordinate transform.

$$\alpha = \arctan\left(\frac{y}{x}\right)$$
$$\beta = \arccos\left(\frac{z}{r}\right)$$
$$r = \sqrt{x^2 + y^2 + z^2}$$

with $\alpha \in [-pi, pi], \beta \in [-\frac{\pi}{2}, \frac{\pi}{2}], r \geq 0$

Parameters

- **x** (*float or array_like*) – x-component of Cartesian coordinates
- **y** (*float or array_like*) – y-component of Cartesian coordinates
- **z** (*float or array_like*) – z-component of Cartesian coordinates

Returns

- **alpha** (float or numpy.ndarray[4]) – Azimuth angle in radians
- **beta** (float or numpy.ndarray[5]) – Elevation angle in radians (with o denoting azimuthal plane)
- **r** (float or numpy.ndarray[6]) – Radius

[1] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
[2] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
[3] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
[4] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
[5] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
[6] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

class sofa.spatial.**Units**

static first_unit(*unit_string*)

static last_unit(*unit_string*)

Metre = 'metre'

Meter = 'meter'

static is_Metre(*value*)

static is_Meter(*value*)

Degree = 'degree'

static is_Degree(*value*)

Radians = 'radians'

static is_Radians(*value*)

static convert_angle_units(*coords*, *dimensions*, *old_units*, *new_units*)

Parameters

- **coords** (*array_like*) – Array of spherical coordinate values
- **dimensions** (*tuple of str*) – Names of the array dimensions in order, must contain "C"
- **old_units** (*str*) – Units of the angle values in the array
- **new_units** (*str*) – Target angle units

Returns **new_coords** (*np.ndarray*) – Array of converted spherical coordinate values in identical dimension order

class sofa.spatial.**System**

Enum of valid coordinate systems

Cartesian = 'cartesian'

Spherical = 'spherical'

static convert(*coords*, *dimensions*, *old_system*, *new_system*, *old_angle_unit=None*, *new_angle_unit=None*)

Parameters

- **coords** (*array_like*) – Array of coordinate values
- **dimensions** (*tuple of str*) – Names of the array dimensions in order, must contain "C"
- **old_system** (*str*) – Coordinate system of the values in the array
- **new_system** (*str*) – Target coordinate system
- **old_angle_unit** (*str, optional*) – Unit of the angular spherical coordinates
- **new_angle_unit** (*str, optional*) – Target unit of the angular spherical coordinates

Returns **new_coords** (*np.ndarray*) – Array of converted coordinate values in identical dimension order

class sofa.spatial.**Coordinates**(*obj*, *descriptor*)

Specialized *sofa.access.Variable* for spatial coordinates

default_values = {'Position': (array([0, 0, 0]), 'cartesian'), 'Up': (array([0, 0, 1]), 'ca

`initialize`(*varies*, *defaults=None*)
Create the variable in the underlying netCDF4 dataset

`Type`
Coordinate system of the values

`get_global_reference_object`()

`get_local_dimension`()

`get_values`(*indices=None*, *dim_order=None*, *system=None*, *angle_unit=None*)
Gets the coordinates in their original reference system

Parameters

- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be returned, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Desired order of dimensions in the output array
- **system** (*str, optional*) – Target coordinate system
- **angle_unit** (*str, optional*) – Unit for spherical angles in the output array

Returns **values** (*np.ndarray*) – Coordinates in the original reference system

`get_global_values`(*indices=None*, *dim_order=None*, *system=None*, *angle_unit=None*)
Transform local coordinates (such as Receiver or Emitter) into the global reference system

Parameters

- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be returned, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Desired order of dimensions in the output array
- **system** (*str, optional*) – Target coordinate system
- **angle_unit** (*str, optional*) – Unit for spherical angles in the output array

Returns **global_values** (*np.ndarray*) – Transformed coordinates in global reference system

`get_relative_values`(*ref_object*, *indices=None*, *dim_order=None*, *system=None*, *angle_unit=None*)
Transform coordinates (such as Receiver or Emitter) into the reference system of a given `sofa.spatial.SpatialObject`, aligning the x-axis with View and the z-axis with Up

Parameters

- **ref_object** (`sofa.spatial.Object`) – Spatial object providing the reference system, None for
- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be returned, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Desired order of dimensions in the output array
- **system** (*str, optional*) – Target coordinate system
- **angle_unit** (*str, optional*) – Unit for spherical angles in the output array

Returns **relative_values** (*np.ndarray*) – Transformed coordinates in original or provided reference system

`set_system`(*ctype=None*, *cunits=None*)
Set the coordinate Type and Units

`set_values`(*values*, *indices=None*, *dim_order=None*, *repeat_dim=None*, *system=None*, *angle_unit=None*)
Sets the coordinate values after converting them to the system and units given by the dataset variable

Parameters

- **values** (*np.ndarray*) – New values for the array range
- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be set, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Dimension names in provided order, regular order assumed
- **repeat_dim** (*tuple of str, optional*) – Tuple of dimension names along which to repeat the values
- **system** (*str, optional*) – Coordinate system of the provided values
- **angle_unit** (*str, optional*) – Angle units of the provided values

`class` sofa.spatial.`SpatialObject`(*database*, *name*)
Spatial object such as Listener, Receiver, Source, Emitter

`Position`
Position of the spatial object relative to its reference system

`View`
View (x-axis) of the spatial object relative to its reference system

`Up`
Up (z-axis) of the spatial object relative to its reference system

`Type`
Coordinate syste, of the values

`initialize`(*fixed=[]*, *variances=[]*, *count=None*)
Create the necessary variables and attributes

Parameters

- **fixed** (*list(str), optional*) – List of spatial coordinates that are fixed for all measurements [“Position”, “View”, “Up”]
- **variances** (*list(str), optional*) – List of spatial coordinates that vary between measurements [“Position”, “View”, “Up”], overrides mentions in fixed
- **count** (*int, optional*) – Number of objects (such as Emitters or Receivers), ignored for Listener and Source

`initialize_coordinates`(*fixed=[]*, *variances=[]*)

Parameters

- **fixed** (*list(str), optional*) – List of spatial coordinates that are fixed for all measurements [“Position”, “View”, “Up”]
- **variances** (*list(str), optional*) – List of spatial coordinates that vary between measurements [“Position”, “View”, “Up”], overrides mentions in fixed

`get_pose`(*indices=None*, *dim_order=None*, *system=None*, *angle_unit=None*)
Gets the spatial object coordinates or their defaults if they have not been defined. Relative

spatial objects return their global pose, or their reference object's pose values if theirs are undefined.

Parameters

- **indices** (*dict(key:str, value:int or slice), optional*) – Key: dimension name, value: indices to be returned, complete axis assumed if not provided
- **dim_order** (*tuple of str, optional*) – Desired order of dimensions in the output arrays
- **system** (*str, optional*) – Target coordinate system
- **angle_unit** (*str, optional*) – Unit for spherical angles in the output arrays

Returns **position, view, up** (*np.ndarray, np.ndarray, np.ndarray*) – Spatial object reference system

3 Version History

Version 0.2.0 (2020-03-03):

- Switched spherical coordinate definition from azimuth and colatitude (0...180) angles to azimuth and elevation (90...-90) to conform to SOFA specifications.
- Provided a more direct access to variable and attribute creation within the dataset
- Reworked the initialization process and DataType creation
- Reworked RoomTypes
- Ensured conventions conform to SOFA 1.0
- Updated usage example

Version 0.1.2 (2020-02-25):

- Fixed issues in relative coordinate access.

Version 0.1.1 (2020-01-23):

- Fixed issues in array accessing and coordinate system conversion.

Version 0.1.0 (2019-07-03):

- Initial release.