sofar Documentation

Release 1.1.2

The pyfar developers

Mar 15, 2024

CONTENTS

sofar

CHAPTER

ONE

README

Sofar is maybe the most complete Python package for the SOFA file format so far. SOFA files store spatially distributed acoustic data such as impulse responses or transfer functions. They are defined by the AES69-2022 standard (see references). These are the key features of sofar

- Uses a complete definition of the AES69-2022 standard (see references) maintained at sofa_conventions
- Read, edit, and write SOFA files
- Add custom attributes to SOFA files
- Full Verification of the content of a SOFA files against AES69-2022
- Upgrade data that uses outdated SOFA conventions
- Open license allows unrestricted use
- sofar is tested using continuous integration on

1.1 Installation

Use pip to install sofar

```
$ pip install sofar
```

(Requires Python >= 3.8)

1.2 Getting Started

Check out read the docs for example use cases a quick introduction to SOFA and sofar, and the complete documentation. A more detailed introduction to SOFA is given by Majdak et. al. 2022 (see references below) Packages related to sofar are listed at pyfar.org. For more information on the SOFA file format visit sofaconventions.org.

1.3 Contributing

Refer to the contribution guidelines for more information.

1.4 References

AES69-2022: *AES standard for file exchange - Spatial acoustic data file format*, Audio Engineering Society, Inc., New York, NY, USA. (https://www.aes.org/publications/standards/search.cfm?docID=99)

P. Majdak, F. Zotter, F. Brinkmann, J. De Muynke, M. Mihocic, and M. Noisternig, "Spatially Oriented Format for Acoustics 2.1: Introduction and Recent Advances", *J. Audio Eng. Soc.*, vol. 70, no. 7/8, pp. 565-584, Jul. 2022. DOI: https://doi.org/10.17743/jaes.2022.0026

CHAPTER

TWO

QUICK TOUR OF SOFA AND SOFAR

If you are new to SOFA and/or sofar, this is a good place to start. SOFA is short for *Spatially Oriented Format for Acoustics* and is an open file format for saving acoustic data, as for example head-related impulse responses (HRIRs). A good places to get more information about SOFA are

- *Documentation of the SOFA conventions*
- The SOFA paper
- sofaconventions.org.
- The SOFA standard AES69-2022

2.1 Creating SOFA objects

To cover a variety of data, SOFA offers different *conventions*. A convention defines, what data can be saved and how it is saved. You should always find the most specific convention for your data. This will help you to identify relevant data and meta data that you should provide along the actual acoustic data. Using sofar, a list of possible conventions can be obtained with

```
import sofar as sf
sf.list_conventions()
```

Let us assume, that you want to store head-related impulse responses (HRIRs). In this case the most specific convention is *SimpleFreeFieldHRIR*. To create a SOFA object use

```
sofa = sf.Sofa("SimpleFreeFieldHRIR")
```

The return value *sofa* is a `sofar.Sofa` object filled with the default values of the *SimpleFreeFieldHRIR* convention. Note that `sf.Sofa()` can also return a sofa object that has only the mandatory attributes. However, it is recommended to start with all attributes and discard empty optional attributes before saving the data.

2.2 Getting information about SOFA objects

To get an overview of the convention, go to the *documentation of the SOFA conventions*.

You might have noted from the documentation that three different kinds of data types can be stored in SOFA files:

- **Attributes:**
  Attributes are meta data stored as strings. There are two kinds of attributes. Global attributes give information about the entire data stored in a SOFA file. All entires starting with *GLOBAL* are such attributes. Specific attributes hold meta data for a certain variable. These attributes thus start with the name of the

variable followed by an underscore, e.g., *ListenerPosition_Units*. An exception to this rule are the data variables, e.g, *Data_IR* is not an attribute but a double variable.

- **Double Variables:**
  Variables of type *double* store numeric data and can be entered as numbers, lists, or numpy arrays.
- **String Variables:**
  Variables of type *string* store strings and can be entered as strings, lists of string, or numpy string arrays.

The data can be mandatory, optional, and read only and must have a shape (dimension in SOFA language) according to the underlying convention. Read on for more information.

To get a quick insight into SOFA objects use

- `sofa.inspect` prints the data stored in a SOFA object or at least gives the shape in case of large arrays that would clutter the output. This is helpful when reading data from an existing SOFA object.
- `sofa.list_dimensions` prints the dimensions of the data inside the SOFA object.
- `sofa.get_dimension` returns the size of a specific dimension.

For the *SimpleFreeFieldHRIR* SOFA object we have the following dimensions

```
sofa.list_dimensions
>>> R = 2 receiver (set by ReceiverPosition of dimension RCI, RCM)
>>> E = 1 emitter (set by EmitterPosition of dimension ECI, ECM)
>>> M = 1 measurements (set by Data_IR of dimension MRN)
>>> N = 1 samples (set by Data_IR of dimension MRN)
>>> C = 3 coordinate dimensions, fixed
>>> I = 1 single dimension, fixed
>>> S = 0 maximum string length
```

In this case, *M* denotes the number of source positions for which HRIRs are available, *R* is the number of ears - which is two - and *N* gives the lengths of the HRIRs in samples. *S* is zero, because the convention does not have any string variables. *C* is always three, because coordinates are either given by x, y, and z values or by their azimuth, elevation and radius in degree.

It is important to be aware of the dimensions and enter data as determined by the convention. SOFA sets the *dimensions* implicitly. This means the dimensions are derived from the data itself, as indicated by the output of `sofa.list_dimensions` above (*set by...*). In some cases, variables can have different shapes. An example for this is the *ReceiverPosition* which can be of shape RCI or RCM. To get a dimension as a variable use

```
sofa.get_dimension("N)
>>> N = 1
```

Let's assume you downloaded a SOFA file from the FABIAN database and want to quickly inspect it. You could use

```
sofa = sf.read_sofa("FABIAN_HRIR_measured_HATO_0.sofa")
sofa.inspect()
>>> GLOBAL_License : Creative Commons (CC-BY). Visit http://creativecommons.org/licenses/
↪by/4.0/ for licence details.
>>> GLOBAL_Organization : Audio Communication Group, TU Berlin, Germany (www.ak.tu-
↪berlin.de)
>>> ReceiverPosition : (R=2, C=3, I=1)
>>>   [[ 0.      0.0662  0.    ]
>>>    [ 0.     -0.0662  0.    ]]
>>> Data_IR : (M=11950, R=2, N=256)
>>> Data_SamplingRate : 44100.0
>>> Data_SamplingRate_Units : hertz
```

Note that the above does not show the entire information for the sake of brevity. This will most likely give you a better idea of the data then looking at the definition of the convention or calling `sofa.list_dimensions`.

2.3 Adding data to SOFA objects

Data can simply be obtained and entered

```
sofa.Data_IR  # prints [0, 0]
sofa.Data_IR = [1, 1]
sofa.SourcePosition = [90, 0, 1.5]
```

Now, the SOFA object contains a single HRIR - which is `1` for the left ear and `1` for the right ear - for a source at `0` degree azimuth, `90` degree elevation and a radius of `1.5` meter. Note that you just entered a list for *Data_IR* although it has to be a three-dimensional double variable. Sofar handles this in two steps.

1. When entering data as lists it is converted to a numpy array with at least two dimensions.
2. Missing dimensions are appended when writing the SOFA object to disk.

You should now fill all mandatory entries of the SOFA object if you were for real. For this example we'll cut it here for the sake of brevity. Let us, however, delete an optional entry that we do not need at this point

```
sofa.delete("SourceUp")
```

In some cases you might want to add custom data - although third party applications most likely won't make use of non-standardized data. Try this to add a temperature value and unit

```
sofa.add_variable("Temperature", 25.1, "double", "MI")
sofa.add_attribute("Temperature_Units", "degree Celsius")
```

After entering the data, the SOFA object should be verified to make sure that your data can (most likely) be read by other applications.

```
sofa.verify()
```

This will check the following

- Are all mandatory data contained?
- Are the names of variables and attributes in accordance with the SOFA standard?
- Are the data types in accordance with the SOFA standard?
- Are the dimensions of the variables consistent and in accordance to the SOFA standard?
- Are the values of attributes consistent and in accordance to the SOFA standard?

If any violations are detected, an error is raised.

2.4 Reading and writing SOFA objects

Note that you usually do not need to call `sofa.verify()` separately because it is by default called if you create write or read a SOFA object. To write your SOFA object to disk type

```
sf.write_sofa("your/path/to/SingleHRIR.sofa", sofa)
```

It is good to know that SOFA files are essentially netCDF4 files which is based on HDF5. They can thus be viewed with HDF View.

To read your sofa file you can use

```
sofa_read = sf.read_sofa("your/path/to/SingleHRIR.sofa")
```

And to see that the written and read files contain the same data you can check

```
sf.equals(sofa, sofa_read)
>>> True
```

2.5 Upgrading SOFA files

SOFA conventions might get updates to fix bugs in the conventions, in case new conventions are introduced, or in case conventions get deprecated. To find out if SOFA data from a file is up to data load it and call

```
sofa.upgrade_convention()
```

which will list upgrade choices or let you know that the convention is already up to date.

2.6 Next steps

For detailed information about sofar refer to the *SOFA objects* and *sofar functions* documentation. For examples on how to work with the data inside SOFA files refer to *Working with SOFA files*.

CHAPTER

THREE

WORKING WITH SOFA FILES

The *Quick tour of SOFA and sofar* showed how to access SOFA files. In many cases you will want to have a closer look at the data inside a SOFA file or use it for further processing. In this section, you will see examples of how to do that using pyfar.

3.1 Retrieving data for specific source and receiver positions

In most cases SOFA files contain data for lots of source or receiver positions and it is often important to get data for a specific position. An elegant way of doing that is to use pyfar Coordinates and Audio objects. Coordinate objects have built in methods to search specific positions and they can convert between a large variety of coordinate systems for you. For audio objects, there is a growing pool of functions for plotting and processing that comes in handy. To use pyfar install it into you python environment

```
$ pip install pyfar
```

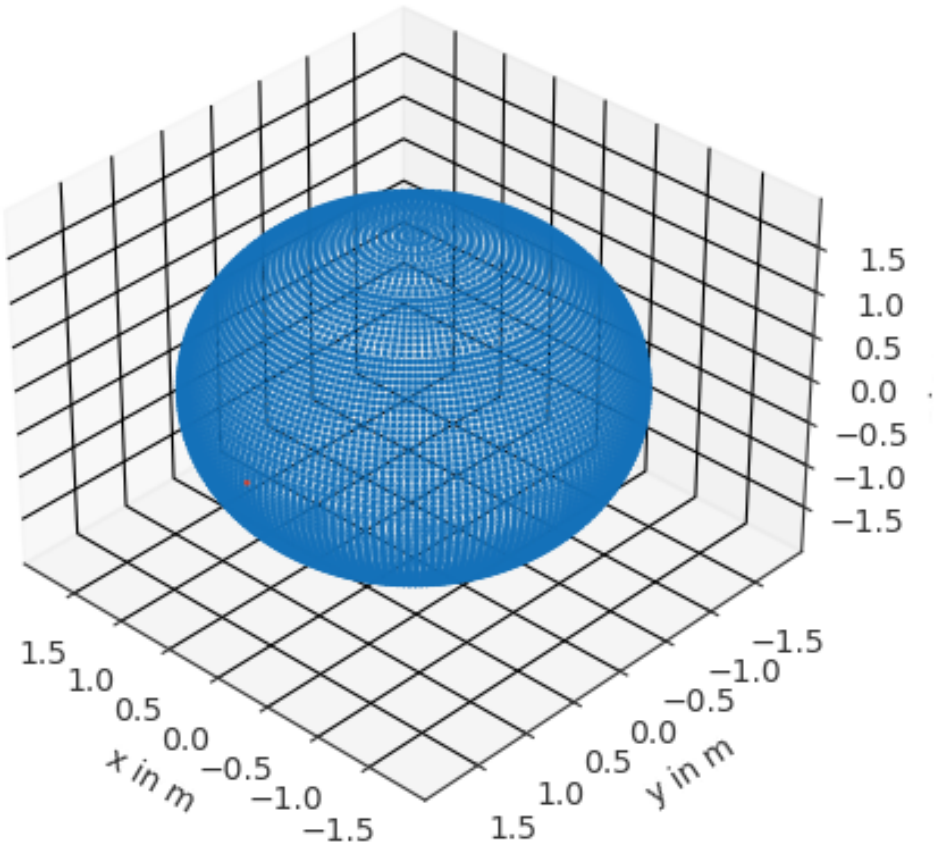
You have to options to get SOFA data into pyfar. The first option is to load the SOFA file with `sofar.read_sofa` and then manually generate Audio and Coordinates objects from the data inside the SOFA file. The second option is to use `pyfar.read_sofa`, which directly returns the Audio and Coordinates objects. Lets be lazy and do that

```
import pyfar as pf
import matplotlib as mpl
import matplotlib.pyplot as plt

data_ir, source_coordinates, receiver_coordinates = pf.io.read_sofa(
    'FABIAN_HRIR_measured_HATO_0.sofa')
```

The SOFA file used in this example is contained head-realated impulse responses (HRIRs) from the FABIAN database. Lets find the HRIR for the source position at the left ear on the horizontal plane. It has an azimuth angle of 90 degrees and an elevation of 0 degrees

```
index, *_ = source_coordinates.find_nearest_k(
    90, 0, 1.5, k=1, domain='sph', convention='top_elev', unit='deg', show=True)
```

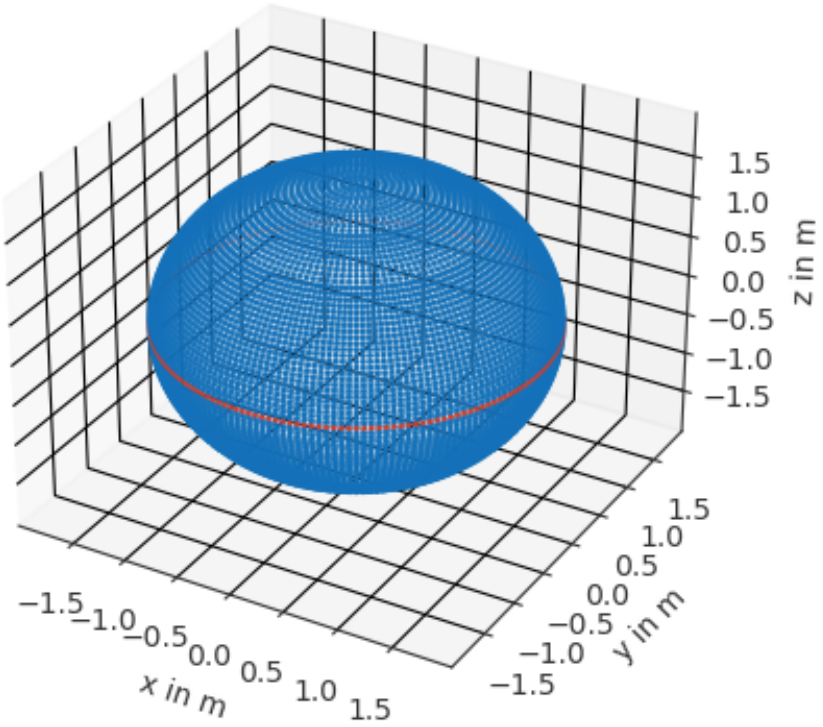
The variable `index = 5930` tells us where to find data for the desired source position. Since we used `show=True` we also get visual feedback for checking if we got the correct source



Note that you get more then the most closest point by using different values for k. It is also possible to all source positions on or in the vicintiy of the horizontal plane using the `find_slice` method of the Coordinates object. Sources on the horizontal plane have zero degree elevation and thus can be obtained by

```
_, mask = source_coordinates.find_slice(
    'elevation', unit='deg', value=0, show=True)
```

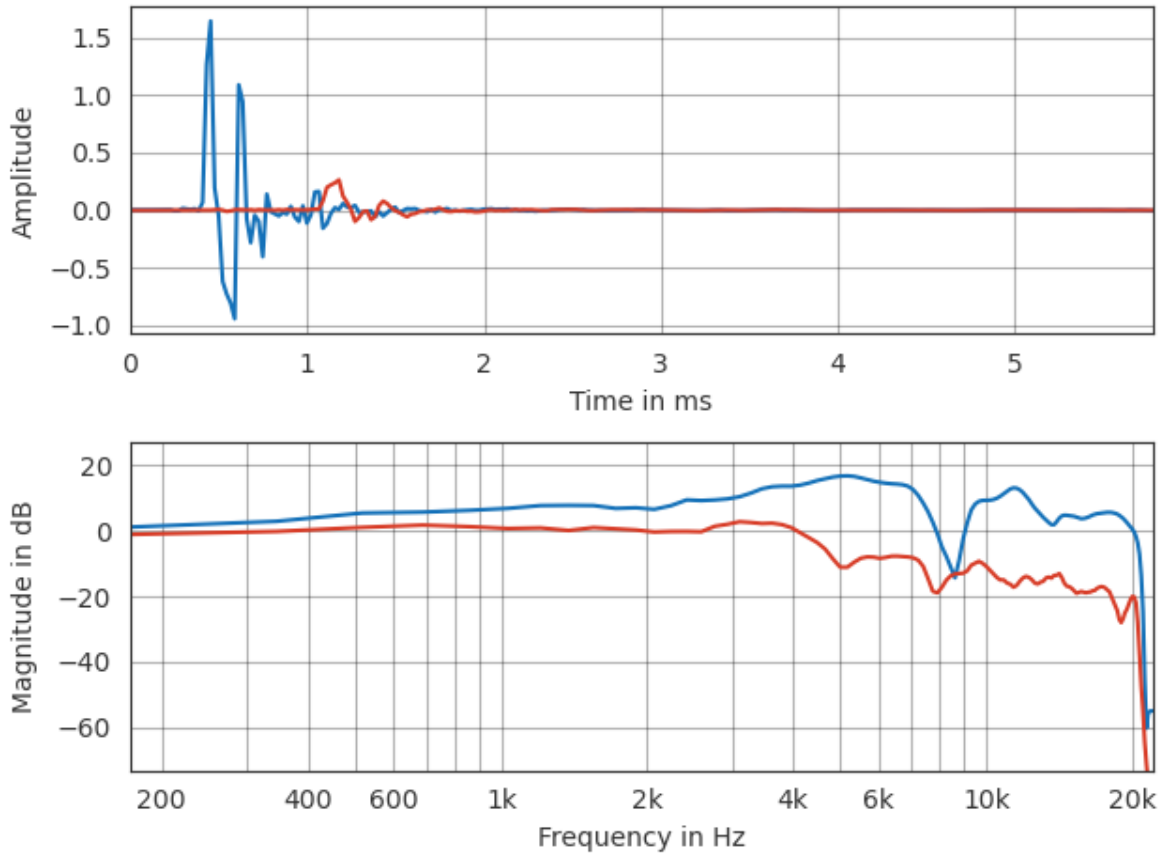
Again, we get visual feedback if we want



3.2 Plotting data

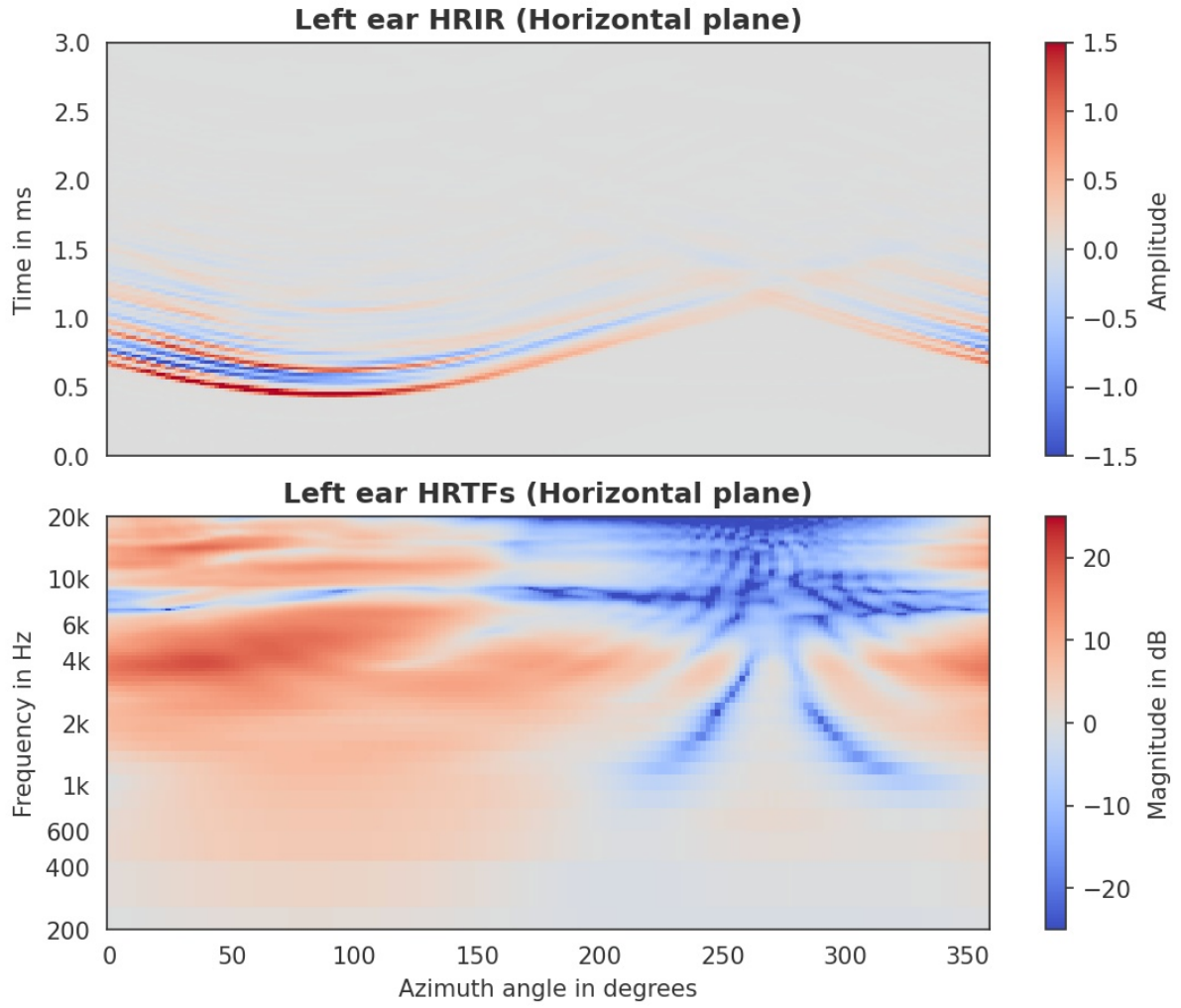
Ploting can be done with the built in plot functions. For example to take a look at the time data and magnitude spectra of a single source position

```
pf.plot.time_freq(data_ir[index])
```



Plotting the entire horizontal plane is also a one liner using `pf.plot.time_freq_2d`, however, a few more lines are required for a nicer formatting

```
with pf.plot.context():

    plt.subplots(2, 1, figsize=(8, 6), sharex=True)

    angles = source_coordinates.get_sph('top_elev', 'deg')[mask, 0]

    ax, qm, cb = pf.plot.time_freq_2d(data_ir[mask, 0], indices=angles,
                                      cmap=mpl.cm.get_cmap(name='coolwarm'))

    ax[0].set_title("Left ear HRIR (Horizontal plane)")
    ax[0].set_xlabel("")
    ax[0].set_ylim(0, 3)
    qm[0].set_clim(-1.5, 1.5)

    ax[1].set_title("Left ear HRTFs (Horizontal plane)")
    ax[1].set_xlabel("Azimuth angle in degrees")
    ax[1].set_ylim(200, 20e3)
    qm[1].set_clim(-25, 25)

    plt.tight_layout
```



3.3 Next steps

For detailed information about sofar refer to the *SOFA objects* and *sofar functions* documentation. Pyfar also offers methods for digital signal processing that wont be detailed here. A god way to dive into that is the pyfar documentation and the pyfar examples notebook.

CHAPTER

FOUR

DOCUMENTATION

4.1 SOFA objects

This section documents sofar SOFA objects. Functions that work on SOFA objects are described in the *sofar functions* guide. For examples on how to use sofar refer to the *Quick tour of SOFA and sofar*.

class sofar.**Sofa**(*convention*, *mandatory=False*, *version='latest'*, *verify=True*)

Bases: `object`

Create a new SOFA object.

Parameters

- **convention** (*str*) – The name of the convention from which the SOFA file is created. See *list_conventions*.
- **mandatory** (*bool, optional*) – If `True`, only the mandatory data of the convention will be returned. The default is `False`, which returns mandatory and optional data.
- **version** (*str, optional*) – The version of the convention as a string, e.g., `' 2.0'`. The default is `'latest'`. Also see *list_conventions*.
- **verify** (*bool, optional*) – Verify the SOFA object by calling *verify*. This helps to find potential errors in the default values and is thus recommended If creating a file does not work, try to call *Sofa* with `verify=False`. The default is `True`.

Returns

sofa – A SOFA object filled with the default values of the convention.

Return type

Sofa

Examples

Create a new SOFA object with default values

```
import sofar as sf

# create SOFA object
sofa = sf.Sofa("SimpleFreeFieldHRIR")
```

Add data as a list

```
sofa.Data_IR = [1, 1]
```

Data can be entered as numbers, numpy arrays or lists. Note the following

1. Lists are converted to numpy arrays with at least two dimensions, i.e., `sofa.Data_IR` is converted to a numpy array of shape (1, 2)
2. Missing dimensions are appended when writing the SOFA object to disk, i.e., `sofa.Data_IR` is written as an array of shape (1, 2, 1) because the SOFA standard AES69-2020 defines it as a three dimensional array with the dimensions (*M: measurements*, *R: receivers*, *N: samples*)
3. When reading data from a SOFA file, array data is always returned as numpy arrays and singleton trailing dimensions are discarded (numpy default). I.e., `sofa.Data_IR` will again be an array of shape (1, 2) after writing and reading to and from disk.
4. One dimensional arrays with only one element will be converted to scalar values. E.g. `sofa.Data_SamplingRate` is stored as an array of shape (1,) inside SOFA files (according to the SOFA standard AES69-2020) but will be a scalar inside SOFA objects after reading from disk.

For more examples refer to the *Quick tour of SOFA and sofar* at https://sofar.readthedocs.io/en/latest/

Methods:

add_attribute(name, value)	Add custom attribute to the SOFA object.
add_missing([mandatory, optional, verbose])	Add missing data with default values.
add_variable(name, value, dtype, dimensions)	Add custom variable to the SOFA object, i.e., numeric or string arrays.
copy()	Return a copy of the SOFA object.
delete(name)	Delete variable or attribute from SOFA object.
get_dimension(dimension)	Get size of a SOFA dimension
info([info])	Print information about the convention of a SOFA object.
inspect([file, issue_handling])	Get information about data inside a SOFA object.
upgrade_convention([target, verify])	Upgrade Sofa data to newer conventions.
verify([issue_handling, mode])	Verify a SOFA object against the SOFA standard.

Attributes:

list_dimensions	Print the dimensions of the SOFA object
protected	If Sofa.protected is `True`, read only data can not be changed.

add_attribute(*name*, *value*)

Add custom attribute to the SOFA object.

Parameters

- **name** (*str*) – Name of the new attribute.
- **value** (*str*) – value to be added.

Examples

```
import sofar as sf
sofa = sf.Sofa("GeneralTF")

# add GLOBAL and Variable attribtue
sofa.add_attribute("GLOBAL_DateMeasured", "8.08.2021")
sofa.add_attribute("Data_Real_Units", "Pascal")
```

add_missing(*mandatory=True*, *optional=True*, *verbose=True*)

Add missing data with default values.

Data might be missing in SOFA objects if the creator did not include it or if a new data was suggested for a newer version of a SOFA convention. Use this function to add the data with its default values.

mandatory
[Bool] Add missing mandatory data. The default is `True`.

optional
[Bool] Add missing optional data. The default is `True`.

verbose
[Bool] Print the information about added data to the console. The default is `True`.

add_variable(*name*, *value*, *dtype*, *dimensions*)

Add custom variable to the SOFA object, i.e., numeric or string arrays.

Parameters

- **name** (*str*) – Name of the new variable.
- **value** (*any*) – value to be added (see *dtype* for restrictions).
- **dtype** (*str*) – Type of the entry to be added in netCDF style:

  'double'
  Use this to store numeric data that can be provided as number list or numpy array.

  'string'
  Use this to store string variables as numpy string arrays of type `'U'` or `'S'`.

- **dimensions** (*str*) – The shape of the new entry as a string. See *list_dimensions*.

Examples

```
import sofar as sf
sofa = sf.Sofa("GeneralTF")

# add numeric data
sofa.add_variable("Temperature", 25.1, "double", "MI")

# add GLOBAL and Variable attribtue
sofa.add_entry(
    "GLOBAL_DateMeasured", "8.08.2021", "attribute", None)
sofa.add_entry(
    "Temperature_Units", "degree Celsius", "attribute", None)
```

(continues on next page)

(continued from previous page)

```
# add a string data
sofa.add_variable(
    "Comment", "Measured with wind screen", "string", "MS")
```

copy()

Return a copy of the SOFA object.

delete(*name*)

Delete variable or attribute from SOFA object.

Note that mandatory data can not be deleted. Check the sofar documentation for a complete list of optional variables and attributes.

Parameters
name (*str*) – Name of the variable or attribute to be deleted

get_dimension(*dimension*)

Get size of a SOFA dimension

SOFA dimensions specify the shape of the data contained in a SOFA object. For a list of all dimensions see *list_dimensions*.

Parameters
dimension (*str*) – The dimension as a string, e.g., 'N'.

Returns
size – the size of the queried dimension.

Return type
int

info(*info='all'*)

Print information about the convention of a SOFA object.

Prints the variable type (attribute, double, string), shape, flags (mandatory, read only) and comment (if any) for each or selected entries.

Parameters
info (*str*) – Specifies the kind of information that is printed:

'all' 'mandatory' 'optional' 'read only' 'data'
Print the name, type, shape, and flags and comment for all or selected entries of the SOFA object. 'data' does not show entries of type attribute.

key
If key is the name of an object attribute, all information for attribute will be printed.

inspect(*file=None*, *issue_handling='print'*)

Get information about data inside a SOFA object.

Prints the values of all attributes and variables with six or less entries and the shapes and type of all numeric and string variables. When printing the values of arrays, single dimensions are discarded for easy of display, i.e., an array of shape (1, 3, 2) will be displayed as an array of shape (3, 2).

Parameters

- **file** (*str*) – Full path of a file under which the information is to be stored in plain text. The default None only print the information to the console.
- **issue_handling** (*str, optional*) – Defines how issues detected during verification of the SOFA object are handeled (see verify)

`'raise'`
Warnings and errors are raised if issues are detected

`'print'`
Issues are printed without raising warnings and errors

`'return'`
Issues are returned as string but neither raised nor printed

`'ignore'`
Issues are ignored, i.e., not raised, printed, or returned.

The default is `print'`.

property list_dimensions

Print the dimensions of the SOFA object

See *inspect* to see the shapes of the data inside the SOFA object and *get_dimension* to get the size/value of a specific dimensions as integer number.

The SOFA file standard defines the following dimensions that are used to define the shape of the data entries:

M
number of measurements

N
number of samles, frequencies, SOS coefficients (depending on self.GLOBAL_DataType)

R
Number of receivers or SH coefficients (depending on ReceiverPosition_Type)

E
Number of emitters or SH coefficients (depending on EmitterPosition_Type)

S
Maximum length of a string in a string array

C
Size of the coordinate dimension. This is always three.

I
Single dimension. This is always one.

property protected

If Sofa.protected is `True`, read only data can not be changed. Only change this to `False` if you know what you are doing, e.g., if you need to repair corrupted SOFA data.

upgrade_convention(*target=None*, *verify=True*)

Upgrade Sofa data to newer conventions.

Calling this with the default arguments returns a list of possible conventions to which the data will be upgraded. If the data is up to date the list will be empty.

Parameters

- **target** (*str, optional*) – The convention and version to which the data should be upgraded as a string. For example `'SimpleFreeFieldHRIR_1.0'` would upgrade the data to the SOFA-Convention *SimpleFreeFieldHRIR* version 1.0. The default is `None` which returns a list of possible conventions to which the data can be updated.
- **verify** (*bool, optional*) – Flag to specify if the data should be verified after the upgrade using *verify*. The default is `True`.

Returns
target – List with available conventions to which the data can be updated. If the data is up to data, the list will be empty. *target* is only returned if *target* is `None`.

Return type
list of strings

`verify`(*issue_handling='raise'*, *mode='write'*)

Verify a SOFA object against the SOFA standard.

This function updates the API, and checks the following

- Are all mandatory data contained? If *issue_handling* is `"raise"` missing mandatory data raises an error. Otherwise mandatory data are added with their default value and a warning is given.
- Are the names of variables and attributes in accordance to the SOFA standard?
- Are the data types in accordance with the SOFA standard?
- Are the dimensions of the variables consistent and in accordance to the SOFA standard?
- Are the values of attributes consistent and in accordance to the SOFA standard?

A detailed set of validation rules can be found at https://github.com/pyfar/sofar/tree/main/sofar/verification_rules

---

Note: *verify* is automatically called when you create a new SOFA object, read a SOFA file from disk, and write a SOFA file to disk (using the default parameters).

The API of a SOFA object consists of four parts, that are stored dictionaries in private attributes. This is required for writing data with `write_sofa` and should usually not be manipulated outside of *verify*

self._convention
The SOFA convention with default values, variable dimensions, flags and comments. These data are read from the official SOFA conventions contained in the SOFA Matlab/Octave API.

self._dimensions
The detected dimensions of the data inside the SOFA object.

self._api
The size of the dimensions (see py:func:*~list_dimensions*). This specifies the dimensions of the data inside the SOFA object.

self._custom
Stores information of custom variables that are not defined by the convention. The format is the same as in *self._convention*.

Parameters

- **issue_handling** (*str, optional*) – Defines how detected issues are handeled

  `'raise'`
  Warnings and errors are raised if issues are detected

  `'print'`
  Issues are printed without raising warnings and errors

  `'return'`
  Issues are returned as string but neither raised nor printed

  The default is `'raise'`.

- **mode** (*str, optional*) – The SOFA standard is more strict for writing data than for reading data.

  'write'
  All units (e.g. 'meter') must be written be lower case.

  'read'
  Units can contain upper case letters (e.g. 'Meter')

  The default is 'write'

Returns
issues – Detected issues as a string. None if no issues were detected. Note that this is only returned if issue_handling='return' (see above)

Return type
str, None

4.2 sofar functions

This section documents general functions of the sofar package. Handling data in SOFA objects is described in *SOFA objects*. For examples on how to use sofa refer to the *Quick tour of SOFA and sofar*. Top-level package for sofar.

Functions:

equals(sofa_a, sofa_b[, verbose, exclude])	Compare two SOFA objects against each other.
list_conventions()	List available SOFA conventions by printing to the console.
read_sofa(filename[, verify, verbose])	Read SOFA file from disk and convert it to SOFA object.
read_sofa_as_netcdf(filename)	Read corrupted SOFA data from disk.
update_conventions([conventions_path, ...])	Update SOFA conventions.
version()	Return version of sofar and SOFA conventions
write_sofa(filename, sofa[, compression])	Write a SOFA object to disk as a SOFA file.

sofar.**equals**(*sofa_a*, *sofa_b*, *verbose=True*, *exclude=None*)

Compare two SOFA objects against each other.

Parameters

- **sofa_a** (Sofa) – SOFA object
- **sofa_b** (Sofa) – SOFA object
- **verbose** (*bool, optional*) – Print differences to the console. The default is True.
- **exclude** (*str, optional*) – Specify what fields should be excluded from the comparison

  'GLOBAL'
  Exclude all global attributes, i.e., fields starting with 'GLOBAL:'

  'DATE'
  Exclude date attributs, i.e., fields that contain 'Date'

  'ATTR'
  Exclude all attributes, i.e., fields that contain ':'

  The default is None, which does not exclude anything.

Returns
is_identical – `True` if sofa_a and sofa_b are identical, `False` otherwise.

Return type
bool

`sofar.list_conventions()`

List available SOFA conventions by printing to the console.

`sofar.read_sofa(filename, verify=True, verbose=True)`

Read SOFA file from disk and convert it to SOFA object.

Numeric data is returned as floats or numpy float arrays unless they have missing data, in which case they are returned as numpy masked arrays.

Parameters

- **filename** (*str*) – The full path to the sofa data.
- **verify** (*bool, optional*) – Verify and update the SOFA object by calling *verify*. This helps to find potential errors in the default values and is thus recommended. If reading a file does not work, try to call *Sofa* with `verify=False`. The default is `True`.
- **verbose** (*bool, optional*) – Print the names of detected custom variables and attributes. The default is `True`

Returns
sofa – Object containing the data from *filename*.

Return type
Sofa

Notes

1. Missing dimensions are appended when writing the SOFA object to disk. E.g., if `sofa.Data_IR` is of shape (1, 2) it is written as an array of shape (1, 2, 1) because the SOFA standard AES69-2020 defines it as a three dimensional array with the dimensions (*M: measurements*, *R: receivers*, *N: samples*)
2. When reading data from a SOFA file, array data is always returned as numpy arrays and singleton trailing dimensions are discarded (numpy default). I.e., `sofa.Data_IR` will again be an array of shape (1, 2) after writing and reading to and from disk.
3. One dimensional arrays with only one element will be converted to scalar values. E.g. `sofa.Data_SamplingRate` is stored as an array of shape (1,) inside SOFA files (according to the SOFA standard AES69-2020) but will be a scalar inside SOFA objects after reading from disk.

`sofar.read_sofa_as_netcdf(filename)`

Read corrupted SOFA data from disk.

Note: *read_sofa_as_netcdf* is intended to read and fix corrupted SOFA data that could not be read by *read_sofa*. The recommend workflow is

- Try to read the data with *read_sofa* and `verify=True`
- If this fails, try the above with `verify=False`
- If this fails, use *read_sofa_as_netcdf*

The SOFA object returned by *read_sofa_as_netcdf* may not work correctly before the issues with the data were fixed, i.e., before the data are in agreement with the SOFA standard AES-69.

Numeric data is returned as floats or numpy float arrays unless they have missing data, in which case they are returned as numpy masked arrays.

Parameters
filename (*str*) – The full path to the NetCDF data.

Returns
sofa – Object containing the data from *filename*.

Return type
Sofa

Notes

1. Missing dimensions are appended when writing the SOFA object to disk. E.g., if `sofa.Data_IR` is of shape (1, 2) it is written as an array of shape (1, 2, 1) because the SOFA standard AES69-2020 defines it as a three dimensional array with the dimensions (*M: measurements*, *R: receivers*, *N: samples*)
2. When reading data from a SOFA file, array data is always returned as numpy arrays and singleton trailing dimensions are discarded (numpy default). I.e., `sofa.Data_IR` will again be an array of shape (1, 2) after writing and reading to and from disk.
3. One dimensional arrays with only one element will be converted to scalar values. E.g. `sofa.Data_SamplingRate` is stored as an array of shape (1,) inside SOFA files (according to the SOFA standard AES69-2020) but will be a scalar inside SOFA objects after reading from disk.

`sofar.update_conventions`(*conventions_path=None*, *assume_yes=False*)

Update SOFA conventions.

SOFA convention define what data is stored in a SOFA file and how it is stored. Updating makes sure that sofar is using the latest conventions. This is done in three steps

1. Download official SOFA conventions as csv files from https://www.sofaconventions.org/conventions/ and https://www.sofaconventions.org/conventions/deprecated/.
2. Convert csv files to json files to be read by sofar.
3. Notify which conventions were newly added or updated.

The csv and json files are stored at sofar/conventions. Sofar works only on the json files. To get a list of all currently available SOFA conventions and their paths see *list_conventions*.

Note: If the official convention contain errors, calling this function might break sofar. If this is the case sofar must be re-installed, e.g., by running `pip install --force-reinstall sofar`. Be sure that you want to do this.

Parameters

- **conventions_path** (*str, optional*) – Path to the folder where the conventions are saved. The default is `None`, which saves the conventions inside the sofar package. Conventions saved under a different path can not be used by sofar. This parameter was added mostly for testing and debugging.
- **response** (*bool, optional*) –

True
Updating the conventions must be confirmed by typing "y".

False
The conventions are updated without confirmation.

The default is True

sofar.**version**()

Return version of sofar and SOFA conventions

sofar.**write_sofa**(*filename: str*, *sofa:* Sofa, *compression=4*)

Write a SOFA object to disk as a SOFA file.

Parameters

- **filename** (*str*) – The filename. '.sofa' is appended to the filename, if it is not explicitly given.
- **sofa** (*object*) – The SOFA object that is written to disk
- **compression** (*int*) – The level of compression with 0 being no compression and 9 being the best compression. The default of 9 optimizes the file size but increases the time for writing files to disk.

Notes

1. Missing dimensions are appended when writing the SOFA object to disk. E.g., if sofa.Data_IR is of shape (1, 2) it is written as an array of shape (1, 2, 1) because the SOFA standard AES69-2020 defines it as a three dimensional array with the dimensions (*M: measurements*, *R: receivers*, *N: samples*)
2. When reading data from a SOFA file, array data is always returned as numpy arrays and singleton trailing dimensions are discarded (numpy default). I.e., sofa.Data_IR will again be an array of shape (1, 2) after writing and reading to and from disk.
3. One dimensional arrays with only one element will be converted to scalar values. E.g. sofa.Data_SamplingRate is stored as an array of shape (1,) inside SOFA files (according to the SOFA standard AES69-2020) but will be a scalar inside SOFA objects after reading from disk.

CHAPTER

FIVE

SOFA CONVENTIONS

5.1 Introduction

SOFA conventions specify what data and metadata must be stored in a SOFA file. Different conventions can be used to store different types of data,e.g., head-related impulse responses or musical instrument directivities.It is advised to always use the conventions that is most specific for thedata.

In the following, SOFA conventions are described in tables with the information

- **Name:** The Name of the data. The prefix *GLOBAL* denotes global attribute, i.e., attributes that pertain the entire data set. Underscores denote attributes that are data specific. E.g., *SourcePosition_Units* denotes the *Units* of the data *SourcePosition*.
- **Type:** The Type of the data.
  - **Attribute:** A verbose description given by a string
  - **Double:** A numeric array of data
  - **String:** A string array of data
- **Default:** The default value
- **Dimensions:** The dimensions of the data. Lower case letters denote the data that sets the dimension.
  - **E:** Number of emitters
  - **R:** Number of receivers
  - **M:** Number of measurements
  - **N:** Number of samples or frequency bins of the data
  - **C:** Number of coordinates (always 3)
  - **I:** Unity dimentions (always 1)
  - **S:** Lengths of the longest string contained in the data (detected automatically)
- **Flags:**
  - **r:** read only data. Data can be written if flag is missing.
  - **m:** mandatory data. Data is optional if flag is missing

5.2 Conventions

- *GeneralTF v1.0*
- *SingleRoomSRIR v1.0*
- *FreeFieldHRIR v1.0*
- *SimpleHeadphoneIR v1.0*
- *SimpleFreeFieldHRIR v1.0*
- *GeneralFIR-E v2.0*
- *SingleRoomMIMOSRIR v1.0*
- *FreeFieldHRTF v1.0*
- *GeneralTF v2.0*
- *FreeFieldDirectivityTF v1.1*
- *SimpleFreeFieldHRSOS v1.0*
- *SimpleFreeFieldHRTF v1.0*
- *GeneralSOS v1.0*
- *SimpleFreeFieldSOS v1.0*
- *GeneralFIR v1.0*
- *GeneralTF-E v1.0*
- *SimpleFreeFieldTF v1.0 (deprecated)*
- *FreeFieldDirectivityTF v1.0 (deprecated)*
- *SimpleHeadphoneIR v0.1 (deprecated)*
- *SingleRoomDRIR v0.2 (deprecated)*
- *GeneralFIRE v1.0 (deprecated)*
- *SimpleHeadphoneIR v0.2 (deprecated)*
- *SimpleFreeFieldTF v0.4 (deprecated)*
- *MultiSpeakerBRIR v0.3 (deprecated)*
- *SingleRoomDRIR v0.3 (deprecated)*
- *SimpleFreeFieldHRIR v0.4 (deprecated)*

5.3 Current

GeneralTF v1.0

This conventions stores TFs for general purposes, i.e., only the mandatory, SOFA general metadata are pre-defined. This convention is based on GeneralFIR.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)	GeneralTF		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)	TF		r, m	We store frequency-dependent data here
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	free field		m	The room information can be arbitrary

continues on next page

Table 1 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[0, 0, 0]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	

continues on next page

Table 1 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	In order to store different directions/positions around the listener, SourcePosition is assumed to vary
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	metre		m	
N (*double*)	0	N	m	Frequency values
N_Longl (*attribute*)	frequency		m	narrative name of N
N_Units (*attribute*)	hertz		m	Unit of the values given in N
Data_Re: (*double*)	0	mRn	m	The real part of the complex spectrum
Data_Im: (*double*)	0	MRN	m	The imaginary part of the complex spectrum

back to top

SingleRoomSRIR v1.0

For measuring SRIRs in a single room with a single excitation source (e.g., a loudspeaker) and a listener containing an arbitrary number of omnidirectional receivers (e.g., a microphone array).

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	SOFA		r, m	
GLOBAl (*attribute*)	2.1		r, m	
GLOBAl (*attribute*)	SingleRoomSRIR		r, m	
GLOBAl (*attribute*)	1.0		r, m	
GLOBAl (*attribute*)	FIR		r, m	Shall be FIR
GLOBAl (*attribute*)	shoebox		m	Shall be ‘shoebox’ or ‘dae’
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)				

continues on next page

Table 2 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	Name of the database. Used for classification of the data.
GLOBAl (*attribute*)				Short name of the Room
GLOBAl (*attribute*)				Informal verbal description of the room
GLOBAl (*attribute*)				Location of the room
GLOBAl (*attribute*)				URI to a file describing the room geometry.
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				

continues on next page

Table 2 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
RoomTe pera- ture (*double*)	0	I, M		Temperature during measurements, given in Kelvin.
RoomTe pera- ture_Uni (*attribute*)	kelvin			Units of the room temperature.
RoomVo ume (*double*)	0	I, M		Volume of the room.
RoomVo ume_Uni (*attribute*)	cubic metre			Units of the room volume.
Room- CornerA (*double*)	[0, 0, 0]	IC, MC		
Room- CornerB (*double*)	[1, 2, 3]	IC, MC		
Room- Corners (*double*)	0	II		The value of this attribute is to be ignored. It only exist to for RoomCorners:Type and Room- Corners:Units
Room- Corners_ (*attribute*)	cartesian			
Room- Corners_ (*attribute*)	metre			
Listen- erPo- sition (*double*)	[0, 0, 0]	MC	m	

continues on next page

Table 2 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView_T (*attribute*)	cartesian		m	
ListenerView_U (*attribute*)	metre		m	
ReceiverDescriptions (*string*)	['']	RS, RSM		R-dependent version of the attribute ReceiverDescription
ReceiverPosition (*double*)	[0, 0, 0]	IC, RCI, RCM	m	
ReceiverPosition_Typ (*attribute*)	spherical		m	Can be of any type enabling both spatially discrete and spatially continuous representations.
ReceiverPosition_Uni (*attribute*)	degree, degree, metre		m	

continues on next page

Table 2 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverVie (*double*)	[1, 0, 0]	RCI, RCM		
ReceiverUp (*double*)	[0, 0, 1]	RCI, RCM		
ReceiverVie (*attribute*)	cartesian			
ReceiverVie (*attribute*)	metre			
SourcePosition (*double*)	[0, 0, 1]	MC	m	
SourcePosition_Typ (*attribute*)	cartesian		m	
SourcePosition_Uni (*attribute*)	metre		m	
SourceView (*double*)	[1, 0, 0]	IC, MC	m	
SourceU (*double*)	[0, 0, 1]	IC, MC	m	
SourceView_Ty (*attribute*)	cartesian		m	
SourceView_Ur (*attribute*)	metre		m	

continues on next page

Table 2 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
EmitterDescriptions (*string*)	['']	ES, ESM		E-dependent version of the attribute EmitterDescription
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
Emitter-Position_Typ (*attribute*)	spherical		m	Shall be 'cartesian' or 'spherical', restricting to spatially discrete emitters.
Emitter-Position_Uni (*attribute*)	degree, degree, metre		m	
Emitter-View (*double*)	[1, 0, 0]	ECI, ECM		
EmitterUp (*double*)	[0, 0, 1]	ECI, ECM		
Emitter-View_Ty (*attribute*)	cartesian			Shall be 'cartesian' or 'spherical', restricting to spatially discrete emitters.
Emitter-View_Ur (*attribute*)	metre			
Measurement-Date (*double*)	0	M		Optional M-dependent date and time of the measurement

continues on next page

Table 2 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Data_IR (*double*)	0	mrn	m	Impulse responses
Data_Sa (*double*)	48000	I, M	m	Sampling rate of the samples in Data.IR and Data.Delay
Data_Sa (*attribute*)	hertz		m	Unit of the sampling rate
Data_De (*double*)	0	IR, MR	m	Additional delay of each IR (in samples)

back to top

FreeFieldHRIR v1.0

An extension of SimpleFreeFieldHRIR in order to consider more complex data sets described in spatially continuous representation. Each HRTF direction corresponds to an emitter, and a consistent measurement for a single listener and all directions is described by a set of the emitter positions surrounding the listener.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	FreeFieldHRIR		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				

continues on next page

Table 3 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	FIR-E		r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	Short name of the listener (as for example the subject ID).
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	Name of the database to which these data belong
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	

continues on next page

Table 3 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	RCI, RCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	Source position is assumed to be the ListenerPosition in order to reflect Emitters surrounding the Listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	IC, ECI, ECM	m	Radius in 'spherical harmonics', Position in 'cartesian' and 'spherical'

continues on next page

Table 3 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Emitter-Position_Typ (*attribute*)	spherical harmonics		m	Can be 'spherical harmonics', 'cartesian', or 'spherical'
Emitter-Position_Uni (*attribute*)	degree, degree, metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
Data_IR (*double*)	[0, 0]	mrne	m	
Data_Sa (*double*)	48000	I, M	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IRI, MRI, MRE	m	Additional delay of each IR (in samples)

back to top

SimpleHeadphoneIR v1.0

Conventions for IRs with a 1-to-1 correspondence between emitter and receiver. The main application for this convention is to store headphone IRs recorded for each emitter and each ear.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	SimpleHeadphoneIR		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	FIR		r, m	We will store IRs here
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	free field		m	Room type is not relevant here

continues on next page

Table 4 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	Correspondence to a database
GLOBAl (*attribute*)			m	Correspondence to a subject from the database
GLOBAl (*attribute*)				Narrative description of the listener (or mannequin)
GLOBAl (*attribute*)				Narrative description of the headphones
GLOBAl (*attribute*)				Name of the headphones manufacturer
GLOBAl (*attribute*)				Name of the headphone model. Must uniquely describe the headphones of the manufacturer
GLOBAl (*attribute*)				URI of the headphone specifications
GLOBAl (*attribute*)			m	Narrative description of the microphones
GLOBAl (*attribute*)			m	Narrative description of the headphone drivers
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 4 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	Default: Headphones are located at the position of the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	eCI, eCM	m	Default: Reflects the correspondence of each emitter to each receiver
EmitterPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 4 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
EmitterPosition_Uni (*attribute*)	metre		m	
SourceManufacturer (*string*)	['']	MS		Optional M-dependent version of the attribute SourceManufacturer
SourceModel (*string*)	['']	MS		Optional M-dependent version of the attribute SourceModel
ReceiverDescriptions (*string*)	['']	MS		R-dependent version of the attribute ReceiverDescription
EmitterDescriptions (*string*)	['']	MS		E-dependent version of the attribute EmitterDescription
MeasurementDate (*double*)	0	M		Optional M-dependent date and time of the measurement
Data_IR (*double*)	[0, 0]	mRn	m	
Data_Sa (*double*)	48000	I, M	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IR, MR	m	

back to top

SimpleFreeFieldHRIR v1.0

This convention set is for HRIRs recorded under free-field conditions or other IRs created under conditions where room information is irrelevant

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	SimpleFreeFieldHRIR		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	FIR		r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	free field		m	

continues on next page

Table 5 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	name of the database to which these data belong
GLOBAl (*attribute*)			m	ID of the subject from the database
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 5 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	Source position is assumed to vary for different directions/positions around the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
EmitterPosition_Typ (*attribute*)	cartesian		m	
EmitterPosition_Uni (*attribute*)	metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	

continues on next page

Table 5 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
SourceU (*double*)	[0, 0, 1]	IC, MC		
SourceView (*double*)	[1, 0, 0]	IC, MC		
SourceView_Ty (*attribute*)	cartesian			
SourceView_Ur (*attribute*)	metre			
Data_IR (*double*)	[0, 0]	mRn	m	
Data_Sa (*double*)	48000	I, M	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IR, MR	m	

back to top

GeneralFIR-E v2.0

This conventions stores IRs for general purposes, i.e., only the mandatory, SOFA general metadata are pre-defined

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	

continues on next page

Table 6 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	GeneralFIR-E		r, m	
GLOBAL (*attribute*)	2.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	FIR-E		r, m	We use FIR datatype which in addition depends on Emitters (E)
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	free field		m	The room information can be arbitrary
GLOBAL (*attribute*)				

continues on next page

Table 6 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[0, 0, 0]	IC, RC, RCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	

continues on next page

Table 6 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	IC, EC, ECM	m	Each speaker is represented as an emitter. Use EmitterPosition to represent the position of a particular speaker. Size of EmitterPosition determines E
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	metre		m	
Data_IR (*double*)	0	mrne	m	Impulse responses
Data_Sa (*double*)	48000	I, M	m	Sampling rate of the samples in Data.IR and Data.Delay
Data_Sa (*attribute*)	hertz		m	Unit of the sampling rate
Data_De (*double*)	0	IRE, MRE	m	Additional delay of each IR (in samples)

back to top

SingleRoomMIMOSRIR v1.0

Single-room multiple-input multiple-output spatial room impulse responses, depending on Emitters

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	SOFA		r, m	
GLOBAl (*attribute*)	2.1		r, m	
GLOBAl (*attribute*)	SingleRoomMIMOSRIR		r, m	
GLOBAl (*attribute*)	1.0		r, m	
GLOBAl (*attribute*)	FIR-E		r, m	Shall be FIR-E
GLOBAl (*attribute*)	shoebox		m	Shall be 'shoebox' or 'dae'
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				

continues on next page

Table 7 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	Name of the database. Used for classification of the data.
GLOBAl (*attribute*)				Short name of the Room
GLOBAl (*attribute*)				Informal verbal description of the room
GLOBAl (*attribute*)				Location of the room
GLOBAl (*attribute*)				URI to a file describing the room geometry.
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				

continues on next page

Table 7 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
RoomTe pera- ture (*dou- ble*)	0	I, M		Temperature during measurements, given in Kelvin.
RoomTe pera- ture_Uni (*at- tribute*)	kelvin			Units of the room temperature
RoomVo ume (*dou- ble*)	0	I, MI		Volume of the room
RoomVo ume_Uni (*at- tribute*)	cubic metre			Units of the room volume
Room- CornerA (*dou- ble*)	[0, 0, 0]	IC, MC		
Room- CornerB (*dou- ble*)	[1, 2, 3]	IC, MC		
Room- Corners (*dou- ble*)	0	II		The value of this attribute is to be ignored. It only exist to for RoomCorners:Type and Room- Corners:Units
Room- Corners_ (*at- tribute*)	cartesian			
Room- Corners_ (*at- tribute*)	metre			
Listen- erPo- sition (*dou- ble*)	[0, 0, 0]	MC	m	

continues on next page

Table 7 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
ReceiverDescriptions (*string*)	['']	RS, RSM		R-dependent version of the attribute ReceiverDescription
ReceiverPosition (*double*)	[0, 0, 0]	IC, RCI, RCM	m	
ReceiverPosition_Typ (*attribute*)	spherical		m	Can be of any type enabling both spatially discrete and spatially continuous representations.
ReceiverPosition_Uni (*attribute*)	degree, degree, metre		m	

continues on next page

Table 7 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverVie (*double*)	[1, 0, 0]	RCI, RCM		
ReceiverUp (*double*)	[0, 0, 1]	RCI, RCM		
ReceiverVie (*attribute*)	cartesian			
ReceiverVie (*attribute*)	metre			
SourcePosition (*double*)	[0, 0, 1]	MC	m	
SourcePosition_Typ (*attribute*)	cartesian		m	
SourcePosition_Uni (*attribute*)	metre		m	
SourceView (*double*)	[1, 0, 0]	IC, MC	m	
SourceU (*double*)	[0, 0, 1]	IC, MC	m	
SourceView_Ty (*attribute*)	cartesian		m	
SourceView_Ur (*attribute*)	metre		m	

continues on next page

Table 7 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
EmitterDescriptions (*string*)	['']	ES, ESM		E-dependent version of the attribute EmitterDescription
EmitterPosition (*double*)	[0, 0, 0]	IC, ECI, ECM	m	Can be of any type enabling both spatially discrete and spatially continuous representations.
Emitter-Position_Typ (*attribute*)	spherical		m	
Emitter-Position_Uni (*attribute*)	degree, degree, metre		m	
Emitter-View (*double*)	[1, 0, 0]	ECI, ECM		
EmitterUp (*double*)	[0, 0, 1]	ECI, ECM		
Emitter-View_Ty (*attribute*)	cartesian			
Emitter-View_Ur (*attribute*)	metre			
Measurement-Date (*double*)	0	M		Optional M-dependent date and time of the measurement.

continues on next page

Table 7 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Data_IR (*double*)	0	mrne	m	Impulse responses
Data_Sa (*double*)	48000	I, M	m	Sampling rate of the samples in Data.IR and Data.Delay
Data_Sa (*attribute*)	hertz		m	Unit of the sampling rate
Data_De (*double*)	0	IRI, MRI, MRE	m	Additional delay of each IR (in samples)

back to top

FreeFieldHRTF v1.0

This conventions is for HRTFs created under conditions where room information is irrelevant and stored as SH coefficients

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	SOFA		r, m	
GLOBAl (*attribute*)	2.1		r, m	
GLOBAl (*attribute*)	FreeFieldHRTF		r, m	
GLOBAl (*attribute*)	1.0		r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	

continues on next page

Table 8 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	TF-E		r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	ID of the subject from the database
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	Name of the database to which these data belong
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Type (*attribute*)	cartesian		m	

continues on next page

Table 8 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	RCI, RCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	Source position is assumed to be the ListenerPosition in order to reflect Emitters surrounding the Listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	IC, ECI, ECM	m	Radius in 'spherical harmonics', Position in 'cartesian' and 'spherical'
EmitterPosition_Typ (*attribute*)	spherical harmonics		m	Can be 'spherical harmonics', 'cartesian', or 'spherical'

continues on next page

Table 8 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Emitter-Position_Uni (*attribute*)	degree, degree, metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
N (*double*)	0	N	m	
N_Longl (*attribute*)	frequency		m	narrative name of N
N_Units (*attribute*)	hertz		m	
Data_Re (*double*)	[0, 0]	mrne	m	
Data_Im (*double*)	[0, 0]	MRNE	m	

back to top

GeneralTF v2.0

This conventions stores TFs for general purposes, i.e., only the mandatory, SOFA general metadata are pre-defined. This convention is based on GeneralFIR.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	GeneralTF		r, m	
GLOBAL (*attribute*)	2.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	TF		r, m	We store frequency-dependent data here
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	free field		m	The room information can be arbitrary

continues on next page

Table 9 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[0, 0, 0]	IC, RC, RCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	

continues on next page

Table 9 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	In order to store different directions/positions around the listener, SourcePosition is assumed to vary
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eC, eCM	m	
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	metre		m	
N (*double*)	0	N	m	Frequency values
N_Longl (*attribute*)	frequency		m	narrative name of N
N_Units (*attribute*)	hertz		m	Unit of the values given in N
Data_Re: (*double*)	0	mrn	m	The real part of the complex spectrum
Data_Im: (*double*)	0	MRN	m	The imaginary part of the complex spectrum

back to top

FreeFieldDirectivityTF v1.1

This conventions stores directivities of acoustic sources (instruments, loudspeakers, singers, talkers, etc) in the frequency domain for multiple musical notes in free field.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	FreeFieldDirectivityTF		r, m	
GLOBAL (*attribute*)	1.1		r, m	
GLOBAL (*attribute*)	TF		r, m	We store frequency-dependent data here
GLOBAL (*attribute*)	free field		m	The room information can be arbitrary, but the spatial setup assumes free field.
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)				

continues on next page

Table 10 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	Name of the database. Used for classification of the data
GLOBAl (*attribute*)				Narrative description of the musician such as position, behavior, or personal data if not data-protected, e.g., 'Christiane Schmidt sitting on the chair', or 'artificial excitation by R2D2'.
GLOBAl (*attribute*)				Narrative description of a measurement. For musical instruments/singers, the note (C1, D1, etc) or the dynamic (pp., ff., etc), or the string played, the playing style (pizzicato, legato, etc.), or the type of excitation (e.g., hit location of a cymbal). For loudspeakers, the system and driver units.
GLOBAl (*attribute*)			m	Narrative description of the acoustic source, e.g., 'Violin', 'Female singer', or '2-way loudspeaker'
GLOBAl (*attribute*)			m	Narrative description of the manufacturer of the source, e.g., 'Stradivari, Lady Blunt, 1721' or 'LoudspeakerCompany'
GLOBAl (*attribute*)				A more detailed structure of the source. In a simple setting, a single Emitter is considered that is collocated with the source. In a more complicated setting, this may be the strings of a violin or the units of a loudspeaker.
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	Position of the microphone array during the measurements.
ListenerPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 10 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition_Uni (*attribute*)	metre		m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	Orientation of the microphone array
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	Up vector of the microphone array
ReceiverPosition (*double*)	[0, 0, 0]	IC, RC, RCM	m	Positions of the microphones during the measurements (relative to the Listener)
ReceiverPosition_Typ (*attribute*)	spherical		m	Type of the coordinate system used.
ReceiverPosition_Uni (*attribute*)	degree, degree, metre		m	Units of the coordinates.
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	Position of the acoustic source (instrument)
SourcePosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 10 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition_Uni (*attribute*)	metre		m	
SourcePosition_Ref (*attribute*)			m	Narrative description of the spatial reference of the source position, e.g., 'The bell' for a trumpet or 'On the front plate between the low- and mid/high-frequency unit' for a loudspeaker. Mandatory in order to provide a reference across different sources.
SourceView (*double*)	[1, 0, 0]	IC, MC	m	View vector for the orientation.
SourceView_Ty (*attribute*)	cartesian		m	
SourceView_Ur (*attribute*)	metre		m	
SourceView_Re (*attribute*)			m	Narrative description of the spatial reference of the source view, e.g., 'Viewing direction of the bell' for a trumpet or 'Perpendicular to the front plate' for a loudspeaker. Mandatory in order to provide a reference across different sources.
SourceU (*double*)	[0, 0, 1]	IC, MC	m	Up vector of the acoustic source (instrument)
SourceU (*attribute*)			m	Narrative description of the spatial reference of the source up, e.g., 'Along the keys, keys up' for a trumpet or 'Perpendicular to the top plate' for a loudspeaker. Mandatory in order to provide a reference across different sources.
EmitterPosition (*double*)	[0, 0, 0]	eC, eCM	m	Position. In a simple settings, a single emitter is considered that is collocated with the source.
EmitterPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 10 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Emitter-Position_Uni (*attribute*)	metre		m	
EmitterDescriptions (*string*)	['']	MS, ES, MES		A more detailed description of the Emitters. For example, this may be the strings of a violin or the units of a loudspeaker.
MIDINo (*double*)	0	I, M		Defines the note played by the source during the measurement. The note is specified a MIDI note by the [https://www.midi.org/specifications-old/item/the-midi-1-0-specification MIDI specifications, version 1.0]. Not mandatory, but recommended for tonal instruments.
Description (*string*)	['']	MS		This variable is used when the description varies with M.
SourceTuningFrequency (*double*)	440	I, M		Frequency (in hertz) to which a musical instrument is tuned to corresponding to the note A4 (MIDINote=69). Recommended for tonal instruments.
N (*double*)	0	N	m	Frequency values
N_Long (*attribute*)	frequency		m	narrative name of N
N_Units (*attribute*)	hertz		m	Units used for N
Data_Re (*double*)	0	mrn	m	Real part of the complex spectrum. The default value 0 indicates that all data fields are initialized with zero values.
Data_Im (*double*)	0	MRN	m	Imaginary part of the complex spectrum

back to top

SimpleFreeFieldHRSOS v1.0

This convention set follows SimpleFreeFieldHRIR but the data is stored as second-order section (SOS) coefficients.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	SimpleFreeFieldHRSOS		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	SOS		r, m	Filters described as second-order section (SOS) coefficients
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	free field		m	

continues on next page

Table 11 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	name of the database to which these data belong
GLOBAl (*attribute*)			m	ID of the subject from the database
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 11 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverPosition_Unit (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	Source position is assumed to vary for different directions/positions around the listener
SourcePosition_Type (*attribute*)	spherical		m	
SourcePosition_Unit (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
EmitterPosition_Type (*attribute*)	cartesian		m	
EmitterPosition_Unit (*attribute*)	metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	

continues on next page

Table 11 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
Data_SO (*double*)	[[[0, 0, 0, 1, 0, 0], [0, 0, 0, 1, 0, 0]]]	mRn	m	Filter coefficients as SOS coefficients.
Data_Sa (*double*)	48000	I, M	m	Sampling rate of the coefficients in Data.SOS and the delay in Data.Delay
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IR, MR	m	Broadband delay (in samples resulting from SamplingRate)

back to top

SimpleFreeFieldHRTF v1.0

This conventions is for HRTFs created under conditions where room information is irrelevant

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	SimpleFreeFieldHRTF		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	

continues on next page

Table 12 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	TF		r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	ID of the subject from the database
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	name of the database to which these data belong

continues on next page

Table 12 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	Source position is assumed to vary for different directions/positions around the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	

continues on next page

Table 12 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
EmitterPosition_Typ (*attribute*)	cartesian		m	
EmitterPosition_Uni (*attribute*)	metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
N (*double*)	0	N	m	
N_Longl (*attribute*)	frequency		m	narrative name of N
N_Units (*attribute*)	hertz		m	
Data_Re (*double*)	[0, 0]	mRn	m	
Data_Im (*double*)	[0, 0]	MRN	m	

back to top

GeneralSOS v1.0

This conventions follows GeneralFIR but the data is stored as second-order section (SOS) coefficients.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	GeneralSOS		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	SOS		r, m	Filters described as second-order section (SOS) coefficients
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				

continues on next page

Table 13 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	free field		m	The room information can be arbitrary
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ListenerView (*double*)	[1, 0, 0]	IC, MC		
ListenerView_ (*attribute*)	cartesian			
ListenerView_ (*attribute*)	metre			
ReceiverPosition (*double*)	[0, 0, 0]	IC, RC, RCM	m	

continues on next page

Table 13 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	In order to store different directions/positions around the listener, SourcePosition is assumed to vary
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
EmitterPosition_Typ (*attribute*)	cartesian		m	
EmitterPosition_Uni (*attribute*)	metre		m	
Data_SO (*double*)	[[[0, 0, 0, 1, 0, 0]]]	mrn	m	Filter coefficients as SOS coefficients.

continues on next page

Table 13 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Data_Sa (*double*)	48000	I, M	m	Sampling rate of the coefficients in Data.SOS and the delay in Data.Delay
Data_Sa (*attribute*)	hertz		m	Unit of the sampling rate
Data_De (*double*)	0	IR, MR	m	Broadband delay (in samples resulting from SamplingRate)

back to top

SimpleFreeFieldSOS v1.0

This convention set follows SimpleFreeFieldHRIR but the data is stored as second-order section (SOS) coefficients.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)	SimpleFreeFieldSOS		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				

continues on next page

Table 14 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	SOS		r, m	Filters described as second-order section (SOS) coefficients
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	name of the database to which these data belong
GLOBAl (*attribute*)			m	ID of the subject from the database
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 14 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	Source position is assumed to vary for different directions/positions around the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
EmitterPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 14 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Emitter-Position_Uni (*attribute*)	metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
Data_SO (*double*)	[[[0, 0, 0, 1, 0, 0], [0, 0, 0, 1, 0, 0]]]	mRn	m	Filter coefficients as SOS coefficients.
Data_Sa (*double*)	48000	I	m	Sampling rate of the coefficients in Data.SOS and the delay in Data.Delay
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IR, MR	m	Broadband delay (in samples resulting from SamplingRate)

back to top

GeneralFIR v1.0

This conventions stores IRs for general purposes, i.e., only the mandatory, SOFA general metadata are pre-defined

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	

continues on next page

Table 15 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	GeneralFIR		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)	FIR		r, m	We store IRs here
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	free field		m	The room information can be arbitrary
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	

continues on next page

Table 15 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[0, 0, 0]	IC, RC, RCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	In order to store different directions/positions around the listener, SourcePosition is assumed to vary
SourcePosition_Typ (*attribute*)	spherical		m	

continues on next page

Table 15 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
EmitterPosition_Typ (*attribute*)	cartesian		m	
EmitterPosition_Uni (*attribute*)	metre		m	
ListenerView (*double*)	[1, 0, 0]	IC, MC		
ListenerView_T (*attribute*)	cartesian			
ListenerView_U (*attribute*)	metre			
Data_IR (*double*)	0	mrn	m	Impulse responses
Data_Sa (*double*)	48000	I, M	m	Sampling rate of the samples in Data.IR and Data.Delay
Data_Sa (*attribute*)	hertz		m	Unit of the sampling rate
Data_De (*double*)	0	IR, MR	m	Additional delay of each IR (in samples)

back to top

GeneralTF-E v1.0

This conventions stores TFs depending in the Emiiter for general purposes, i.e., only the mandatory, SOFA general metadata are pre-defined. This convention is based on GeneralTF

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	SOFA		r, m	
GLOBAl (*attribute*)	2.1		r, m	
GLOBAl (*attribute*)	GeneralTF-E		r, m	
GLOBAl (*attribute*)	1.0		r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	TF-E		r, m	We store frequency-dependent data depending on the emitter here
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				

continues on next page

Table 16 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	free field		m	The room information can be arbitrary
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[0, 0, 0]	IC, RC, RCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	

continues on next page

Table 16 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	In order to store different directions/positions around the listener, SourcePosition is assumed to vary
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	IC, EC, ECM	m	
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	metre		m	
N (*double*)	0	N	m	Frequency values
N_Longl (*attribute*)	frequency		m	narrative name of N
N_Units (*attribute*)	hertz		m	Unit of the values given in N
Data_Re: (*double*)	0	mrne	m	The real part of the complex spectrum
Data_Im: (*double*)	0	MRNE	m	The imaginary part of the complex spectrum

back to top

5.4 Deprecated

SimpleFreeFieldTF v1.0

This convention is deprecated. Use *SimpleFreeFieldHRTF_1.0* instead.

This conventions is for TFs created under conditions where room information is irrelevant

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)	SimpleFreeFieldTF		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	TF		r, m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)	No license provided, ask the author for permission		m	
GLOBAL (*attribute*)			m	ID of the subject from the database

continues on next page

Table 17 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	name of the database to which these data belong
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	

continues on next page

Table 17 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	Source position is assumed to vary for different directions/positions around the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
EmitterPosition_Typ (*attribute*)	cartesian		m	
EmitterPosition_Uni (*attribute*)	metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	

continues on next page

Table 17 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
N (*double*)	0	N	m	
N_Longl (*attribute*)	frequency			
N_Units (*attribute*)	hertz			
Data_Re: (*double*)	[0, 0]	mRn	m	
Data_Im: (*double*)	[0, 0]	MRN	m	

back to top

FreeFieldDirectivityTF v1.0

This convention is deprecated. Use *FreeFieldDirectivityTF_1.1* instead.

This conventions stores directivities of acoustic sources (instruments, loudspeakers, singers, talkers, etc) in the frequency domain for multiple musical notes in free field.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	2.1		r, m	
GLOBAL (*attribute*)	FreeFieldDirectivityTF		r, m	

continues on next page

Table 18 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	1.0		r, m	
GLOBAl (*attribute*)	TF		r, m	We store frequency-dependent data here
GLOBAl (*attribute*)	free field		m	The room information can be arbitrary, but the spatial setup assumes free field.
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				

continues on next page

Table 18 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	Name of the database. Used for classification of the data
GLOBAl (*attribute*)				Narrative description of the musician such as position, behavior, or personal data if not data-protected, e.g., 'Christiane Schmidt sitting on the chair', or 'artificial excitation by R2D2'.
GLOBAl (*attribute*)				Narrative description of a measurement. For musical instruments/singers, the note (C1, D1, etc) or the dynamic (pp., ff., etc), or the string played, the playing style (pizzicato, legato, etc.), or the type of excitation (e.g., hit location of a cymbal). For loudspeakers, the system and driver units.
GLOBAl (*attribute*)			m	Narrative description of the acoustic source, e.g., 'Violin', 'Female singer', or '2-way loudspeaker'
GLOBAl (*attribute*)			m	Narrative description of the manufacturer of the source, e.g., 'Stradivari, Lady Blunt, 1721' or 'LoudspeakerCompany'
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	Position of the microphone array during the measurements.
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	Orientation of the microphone array
ListenerView_T (*attribute*)	cartesian		m	
ListenerView_U (*attribute*)	metre		m	

continues on next page

Table 18 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	Up vector of the microphone array
ReceiverPosition (*double*)	[0, 0, 1]	IC, RC, RCM	m	Positions of the microphones during the measurements (relative to the Listener)
ReceiverPosition_Typ (*attribute*)	spherical		m	
ReceiverPosition_Uni (*attribute*)	degree, degree, metre		m	
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	Position of the acoustic source (instrument)
SourcePosition_Typ (*attribute*)	cartesian		m	
SourcePosition_Uni (*attribute*)	metre		m	
SourcePosition_Ref (*attribute*)			m	Narrative description of the spatial reference of the source position, e.g., for the trumpet, 'The bell'. Mandatory in order to provide a reference across different instruments
SourceView (*double*)	[1, 0, 0]	IC, MC	m	Orientation of the acoustic source (instrument)

continues on next page

Table 18 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Source-View_Ty (*attribute*)	cartesian		m	
Source-View_Ur (*attribute*)	metre		m	
Source-View_Re (*attribute*)			m	Narrative description of the spatial reference of the source view, e.g., for the trumpet, 'Viewing direction of the bell'. Mandatory in order to provide a reference across different instruments
SourceU] (*double*)	[0, 0, 1]	IC, MC	m	Up vector of the acoustic source (instrument)
SourceU] (*attribute*)			m	Narrative description of the spatial reference of the source up, e.g., for the trumpet, 'Along the keys, keys up'. Mandatory in order to provide a reference across different instruments
Emitter-Position (*double*)	[0, 0, 0]	IC, MC	m	A more detailed structure of the Source. In a simple settings, a single Emitter is considered that is collocated with the source.
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	metre		m	
EmitterDescription (*string*)	['']	IS, MS		A more detailed structure of the source. In a simple setting, a single Emitter is considered that is collocated with the source. In a more complicated setting, this may be the strings of a violin or the units of a loudspeaker.
MIDINot (*double*)	0	I, M		Defines the note played by the source during the measurement. The note is specified a MIDI note by the [https://www.midi.org/specifications-old/item/the-midi-1-0-specification MIDI specifications, version 1.0]. Not mandatory, but recommended for tonal instruments.

continues on next page

Table 18 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Description (*string*)	['']	MS		This variable is used when the description varies with M.
SourceTuningFrequency (*double*)	440	I, M		Frequency (in hertz) to which a musical instrument is tuned to corresponding to the note A4 (MIDINote=69). Recommended for tonal instruments.
N (*double*)	0	N	m	Frequency values
N_Longl (*attribute*)	frequency		m	
N_Units (*attribute*)	hertz		m	Units used for N
Data_Re (*double*)	0	mrn	m	Real part of the complex spectrum. The default value 0 indicates that all data fields are initialized with zero values.
Data_Im (*double*)	0	MRN	m	Imaginary part of the complex spectrum

back to top

SimpleHeadphoneIR v0.1

This convention is deprecated. Use *SimpleHeadphoneIR_1.0* instead.

Conventions for IRs with a 1-to-1 correspondence between emitter and receiver. The main application for this convention is to store headphone IRs recorded for each emitter and each ear.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	SOFA		r, m	
GLOBAl (*attribute*)	1.0		r, m	
GLOBAl (*attribute*)	SimpleHeadphoneIR		r, m	
GLOBAl (*attribute*)	0.1		r, m	

continues on next page

Table 19 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	FIR		r, m	We will store IRs here
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	Room type is not relevant here
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	

continues on next page

Table 19 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	Correspondence to a database
GLOBAl (*attribute*)			m	Correspondence to a subject from the database
GLOBAl (*attribute*)			m	Narrative description of the listener (or mannequin)
GLOBAl (*attribute*)			m	Narrative description of the headphones
GLOBAl (*attribute*)			m	Name of the headphones manufacturer
GLOBAl (*attribute*)			m	Name of the headphone model. Must uniquely describe the headphones of the manufacturer
GLOBAl (*attribute*)			m	URI of the headphone specifications
GLOBAl (*attribute*)			m	Narrative description of the microphones
GLOBAl (*attribute*)			m	Narrative description of the headphone drivers
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	meter		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	

continues on next page

Table 19 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	meter		m	
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	Default: Headphones are located at the position of the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, meter		m	
EmitterPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	eCI, eCM	m	Default: Reflects the correspondence of each emitter to each receiver
EmitterPosition_Typ (*attribute*)	cartesian		m	
EmitterPosition_Uni (*attribute*)	meter		m	
SourceManufacturer (*string*)	['']	MS		Optional M-dependent version of the attribute SourceManufacturer

continues on next page

Table 19 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourceModel (*string*)	['']	MS		Optional M-dependent version of the attribute SourceModel
ReceiverDescription (*string*)	['']	MS		Optional M-dependent version of the attribute ReceiverDescription
EmitterDescription (*string*)	['']	MS		Optional M-dependent version of the attribute EmitterDescription
MeasurementDate (*double*)	0	M		Optional M-dependent date and time of the measurement
Data_IR (*double*)	[1, 1]	mRn	m	
Data_Sa (*double*)	48000	I	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IR, MR	m	

back to top

SingleRoomDRIR v0.2

This convention is deprecated. Use *SingleRoomSRIR_1.0* instead.

This convention stores arbitrary number of receivers while providing an information about the room. The main application is to store DRIRs for a single room.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	

continues on next page

Table 20 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	SingleRoomDRIR		r, m	
GLOBAl (*attribute*)	0.2		r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	FIR		r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	reverberant		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	

continues on next page

Table 20 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	meter		m	
ReceiverPosition (*double*)	[0, 0, 0]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	meter		m	

continues on next page

Table 20 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	
SourcePosition_Typ (*attribute*)	cartesian		m	
SourcePosition_Uni (*attribute*)	meter		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	meter		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_T (*attribute*)	cartesian		m	
ListenerView_U (*attribute*)	metre		m	

continues on next page

Table 20 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourceU] (*double*)	[0, 0, 1]	IC, MC	m	
Source-View (*double*)	[-1, 0, 0]	IC, MC	m	
Source-View_Ty (*attribute*)	cartesian		m	
Source-View_Ur (*attribute*)	metre		m	
Data_IR (*double*)	[1]	mRn	m	
Data_Sa (*double*)	48000	I	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0]	IR, MR	m	

back to top

GeneralFIRE v1.0

This convention is deprecated. Use *GeneralFIR-E_2.0* instead.

This conventions stores IRs for general purposes, i.e., only the mandatory, SOFA general metadata are pre-defined

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)	GeneralFIRE		r, m	
GLOBAL (*attribute*)	1.0		r, m	

continues on next page

Table 21 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	FIRE		r, m	We use FIR datatype which in addition depends on Emitters (E)
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	The room information can be arbitrary
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	

continues on next page

Table 21 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[0, 0, 0]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	

continues on next page

Table 21 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	Each speaker is represented as an emitter. Use EmitterPosition to represent the position of a particular speaker. Size of EmitterPosition determines E
EmitterPosition_Typ (*attribute*)	cartesian		m	
EmitterPosition_Uni (*attribute*)	metre		m	
Data_IR (*double*)	0	mREn	m	Impulse responses
Data_Sa (*double*)	48000	I	m	Sampling rate of the samples in Data.IR and Data.Delay
Data_Sa (*attribute*)	hertz		m	Unit of the sampling rate
Data_De (*double*)	0	IRE, MRE	m	Additional delay of each IR (in samples)

back to top

SimpleHeadphoneIR v0.2

This convention is deprecated. Use *SimpleHeadphoneIR_1.0* instead.

Conventions for IRs with a 1-to-1 correspondence between emitter and receiver. The main application for this convention is to store headphone IRs recorded for each emitter and each ear.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)	SimpleHeadphoneIR		r, m	

continues on next page

Table 22 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	0.2		r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	FIR		r, m	We will store IRs here
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	Room type is not relevant here
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	

continues on next page

Table 22 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	Correspondence to a database
GLOBAl (*attribute*)			m	Correspondence to a subject from the database
GLOBAl (*attribute*)			m	Narrative description of the listener (or mannequin)
GLOBAl (*attribute*)			m	Narrative description of the headphones
GLOBAl (*attribute*)			m	Name of the headphones manufacturer
GLOBAl (*attribute*)			m	Name of the headphone model. Must uniquely describe the headphones of the manufacturer
GLOBAl (*attribute*)			m	URI of the headphone specifications
GLOBAl (*attribute*)			m	Narrative description of the microphones
GLOBAl (*attribute*)			m	Narrative description of the headphone drivers
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	

continues on next page

Table 22 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	Default: Headphones are located at the position of the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	eCI, eCM	m	Default: Reflects the correspondence of each emitter to each receiver
EmitterPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 22 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
EmitterPosition_Uni (*attribute*)	metre		m	
SourceManufacturer (*string*)	['']	MS		Optional M-dependent version of the attribute SourceManufacturer
SourceModel (*string*)	['']	MS		Optional M-dependent version of the attribute SourceModel
ReceiverDescription (*string*)	['']	MS		Optional M-dependent version of the attribute ReceiverDescription
EmitterDescription (*string*)	['']	MS		Optional M-dependent version of the attribute EmitterDescription
MeasurementDate (*double*)	0	M		Optional M-dependent date and time of the measurement
Data_IR (*double*)	[0, 0]	mRn	m	
Data_Sa (*double*)	48000	I	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IR, MR	m	

back to top

SimpleFreeFieldTF v0.4

This convention is deprecated. Use *SimpleFreeFieldHRTF_1.0* instead.

This conventions is for TFs created under conditions where room information is irrelevant

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)	SOFA		r, m	
GLOBAl (*attribute*)	1.0		r, m	
GLOBAl (*attribute*)	SimpleFreeFieldTF		r, m	
GLOBAl (*attribute*)	0.4		r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	TF		r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	ID of the subject from the database
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				

continues on next page

Table 23 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	free field		m	
GLOBAL (*attribute*)				
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			m	
GLOBAL (*attribute*)			m	name of the database to which these data belong
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	meter		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 23 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ReceiverPosition_Uni (*attribute*)	meter		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	Source position is assumed to vary for different directions/positions around the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, meter		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	meter		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	

continues on next page

Table 23 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	meter		m	
N (*double*)	0	N	m	
N_Longl (*attribute*)	frequency			
N_Units (*attribute*)	hertz			
Data_Re (*double*)	[1, 1]	mRn	m	
Data_Im (*double*)	[0, 0]	MRN	m	

back to top

MultiSpeakerBRIR v0.3

This convention is deprecated. Use *SingleRoomMIMOSRIR_1.0* instead.

This convention is for BRIRs recorded in reverberant conditions from multiple loudspeaker sources at a number of listener orientations.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)	MultiSpeakerBRIR		r, m	
GLOBAL (*attribute*)	0.3		r, m	
GLOBAL (*attribute*)			r, m	

continues on next page

Table 24 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	FIRE		r, m	We use FIR datatype which in addition depends on Emitters (E)
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	reverberant		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	name of the database to which these data belong

continues on next page

Table 24 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	ID of the subject from the database
GLOBAl (*attribute*)				narrative description of the room
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	
SourcePosition_Typ (*attribute*)	spherical		m	

continues on next page

Table 24 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition_Uni (*attribute*)	degree, degree, metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	Each speaker is represented as an emitter. Use EmitterPosition to represent the position of a particular speaker. Size of EmitterPosition determines E
EmitterPosition_Typ (*attribute*)	cartesian		m	
EmitterPosition_Uni (*attribute*)	metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
EmitterUp (*double*)	[0, 0, 1]	ECI, ECM		When EmitterUp provided, EmitterView must be provided as well
EmitterView (*double*)	[1, 0, 0]	ECI, ECM		When EmitterView provided, EmitterUp must be provided as well

continues on next page

Table 24 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
EmitterView_Ty (*attribute*)	cartesian			
EmitterView_Ur (*attribute*)	metre			
Data_IR (*double*)	[1, 1]	mREn	m	
Data_Sa (*double*)	48000	I	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IRE, MRE	m	

back to top

SingleRoomDRIR v0.3

This convention is deprecated. Use *SingleRoomSRIR_1.0* instead.

This convention stores arbitrary number of receivers while providing an information about the room. The main application is to store DRIRs for a single room.

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)	SingleRoomDRIR		r, m	
GLOBAL (*attribute*)	0.3		r, m	
GLOBAL (*attribute*)			r, m	

continues on next page

Table 25 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)	FIR		r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	reverberant		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	

continues on next page

Table 25 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)			m	
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	metre		m	
ReceiverPosition (*double*)	[0, 0, 0]	RCI, RCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	metre		m	
SourcePosition (*double*)	[0, 0, 0]	IC, MC	m	
SourcePosition_Typ (*attribute*)	cartesian		m	

continues on next page

Table 25 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
SourcePosition_Uni (*attribute*)	metre		m	
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	metre		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_ (*attribute*)	cartesian		m	
ListenerView_ (*attribute*)	metre		m	
SourceU (*double*)	[0, 0, 1]	IC, MC	m	
SourceView (*double*)	[-1, 0, 0]	IC, MC	m	
SourceView_Ty (*attribute*)	cartesian		m	

continues on next page

Table 25 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
Source-View_Ur (*attribute*)	metre		m	
Data_IR (*double*)	[0]	mrn	m	
Data_Sa (*double*)	48000	I	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0]	IR, MR	m	

back to top

SimpleFreeFieldHRIR v0.4

This convention is deprecated. Use *SimpleFreeFieldHRIR_1.0* instead.

This convention set is for HRIRs recorded under free-field conditions or other IRs created under conditions where room information is irrelevant

Name (Type)	Default	Dim.	Flags	Comment
GLOBAL (*attribute*)	SOFA		r, m	
GLOBAL (*attribute*)	1.0		r, m	
GLOBAL (*attribute*)	SimpleFreeFieldHRIR		r, m	
GLOBAL (*attribute*)	0.4		r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)			r, m	
GLOBAL (*attribute*)				

continues on next page

Table 26 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	FIR		r, m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	No license provided, ask the author for permission		m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)	free field		m	
GLOBAl (*attribute*)				
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	
GLOBAl (*attribute*)			m	name of the database to which these data belong
GLOBAl (*attribute*)			m	ID of the subject from the database

continues on next page

Table 26 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
ListenerPosition (*double*)	[0, 0, 0]	IC, MC	m	
ListenerPosition_Typ (*attribute*)	cartesian		m	
ListenerPosition_Uni (*attribute*)	meter		m	
ReceiverPosition (*double*)	[[0, 0.09, 0], [0, -0.09, 0]]	rCI, rCM	m	
ReceiverPosition_Typ (*attribute*)	cartesian		m	
ReceiverPosition_Uni (*attribute*)	meter		m	
SourcePosition (*double*)	[0, 0, 1]	IC, MC	m	Source position is assumed to vary for different directions/positions around the listener
SourcePosition_Typ (*attribute*)	spherical		m	
SourcePosition_Uni (*attribute*)	degree, degree, meter		m	

continues on next page

Table 26 – continued from previous page

Name (Type)	Default	Dim.	Flags	Comment
EmitterPosition (*double*)	[0, 0, 0]	eCI, eCM	m	
Emitter-Position_Typ (*attribute*)	cartesian		m	
Emitter-Position_Uni (*attribute*)	meter		m	
ListenerUp (*double*)	[0, 0, 1]	IC, MC	m	
ListenerView (*double*)	[1, 0, 0]	IC, MC	m	
ListenerView_T (*attribute*)	cartesian		m	
ListenerView_U (*attribute*)	meter		m	
Data_IR (*double*)	[1, 1]	mRn	m	
Data_Sa (*double*)	48000	I	m	
Data_Sa (*attribute*)	hertz		m	
Data_De (*double*)	[0, 0]	IR, MR	m	

back to top

CHAPTER

SIX

CONTRIBUTING

6.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1.1 Types of Contributions

Report Bugs and Submit Feedback

The best way to report bugs of send feedback is to open an issue at https://github.com/pyfar/sofar/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Fix Bugs or Implement Features

Look through the GitHub issues for bugs. Anything tagged with “bug” or “enhancement” is open to whoever wants to implement it. It might be good to contact us first, to see if anyone is already working on it.

Write Documentation

sofar could always use more documentation, whether as part of the official sofar docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.2 Get Started!

Ready to contribute? Here's how to set up *sofar* for local development.

1. Fork the *sofar* repo on GitHub.
2. Clone your fork locally and cd into the sofar directory:

```
$ git clone --recursive https://github.com/pyfar/sofar.git
$ cd sofar/
```

3. Note that some graphical Git interfaces can not do the recursive clone. If the folder sofar/sofa_conventions is empty try

   $ git submodule update –init

4. Install your local copy into a virtualenv. Assuming you have Anaconda or Miniconda installed, this is how you set up your fork for local development:

```
$ conda create --name sofar python
$ conda activate sofar
$ conda install pip
$ pip install -e .
$ pip install -r requirements_dev.txt
```

5. Create a branch for local development. Indicate the intention of your branch in its respective name (i.e. *feature/branch-name* or *bugfix/branch-name*):

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass flake8 and the tests:

```
$ flake8 sofar tests
$ pytest
```

   flake8 test must pass without any warnings for *./sofar* and *./tests* using the default or a stricter configuration. Flake8 ignores *E123/E133, E226* and *E241/E242* by default. If necessary adjust the your flake8 and linting configuration in your IDE accordingly.

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

6.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 3.7 and 3.8. Check https://travis-ci.com/pyfar/sofar/pull_requests and make sure that the tests pass for all supported Python versions.

6.1.4 Testing Guidelines

Sofar uses test-driven development based on three steps and continuous integration to test and monitor the code. In the following, you'll find a guideline. Note: these instructions are not generally applicable outside of sofar.

- The main tool used for testing is pytest.
- All tests are located in the *tests/* folder.
- Make sure that all important parts of sofar are covered by the tests. This can be checked using *coverage* (see below).
- In case of sofar, mainly **state verification** is applied in the tests. This means that the outcome of a function is compared to a desired value (`assert ...`). For more information, it is refered to Martin Fowler's article.

Tips

Pytest provides several, sophisticated functionalities which could reduce the effort of implementing tests.

- Similar tests executing the same code with different variables can be parametrized.
- Feel free to add more recommendations on useful pytest functionalities here. Consider, that a trade-off between easy implemention and good readability of the tests needs to be found.

You can create an html report on the test coverage by calling

    $ pytest –cov=. –cov-report=html

6.1.5 Writing the Documentation

Sofar follows the numpy style guide for the docstring. A docstring has to consist at least of

- A short and/or extended summary,
- the Parameters section, and
- the Returns section

Optional fields that are often used are

- References,
- Examples, and
- Notes

Here are a few tips to make things run smoothly

- Use the tags `:py:func:`, `:py:mod:`, and `:py:class:` to reference sofar functions, modules, and classes: For example `:py:func:`~sofar.write_sofa`` for a link that displays only the function name.
- Code snippets and values as well as external modules, classes, functions are marked by double ticks `` to appear in mono spaced font, e.g., `x=3` or `sofar.Signal`.
- Parameters, returns, and attributes are marked by single ticks ` to appear as emphasized text, e.g., *unit*.
- Use `[#]_` and `.. [#]` to get automatically numbered footnotes.
- Do not use footnotes in the short summary. Only use footnotes in the extended summary if there is a short summary. Otherwise, it messes with the auto-footnotes.

See the Sphinx homepage for more information.

6.1.6 Building the Documentation

You can build the documentation of your branch using Sphinx by executing the make script inside the docs folder.

```
$ cd docs/
$ make html
```

After Sphinx finishes you can open the generated html using any browser

```
$ docs/_build/index.html
```

Note that some warnings are only shown the first time you build the documentation. To show the warnings again use

```
$ make clean
```

before building the documentation.

Submodules

To update the submodule containing the conventions and verification rules run

$ git submodule update –init –recursive $ git submodule update –recursive –remote

and then commit the changes

Deploying

A reminder for the maintainers on how to deploy.

- Commit all changes to develop
- Update HISTORY.rst in develop
- Check if new contributors should be added to AUTHORS.rst
- Merge develop into main

Switch to main and run:

```
$ bumpversion patch --verbose # possible: major / minor / patch
```

Bumpversion will update all version strings, create and commit tags by default

$ git push –follow-tags

Continuous integration will then deploy to PyPI if tests pass.

- Merge main back into develop

CHAPTER

SEVEN

OTHER

7.1 Credits

7.1.1 The sofar developers

- Fabian Brinkmann
- Marco Berzborn

7.1.2 Funding

- Institute for Advanced Procrastination

7.2 History

7.2.1 1.1.2 (2024-2-22)

- Fix for working with rye package manager (PR #75)
- Add testing for Python 3.12 (PR #76)

7.2.2 1.1.1 (2023-7-7)

- Fix deploying to PyPi.org

7.2.3 1.1.0 (2023-7-7)

- Deprecate FreeFieldDirectivityTV 1.0 in favor of FreeFieldDirectivityTV 1.1 (according to sofaconventoins.org and AES69-2022)
- Add *sofar.read_sofa_as_netcdf* for reading SOFA files with erroneous data
- Document SOFA conventions on https://sofar.readthedocs.io/en/stable/resources/conventions.html. *Sofa.info()* will this be deprecated in sofar v1.3.0
- *sofar.read_sofa* and *sofar.write_sofa* now accept filenames and path objects
- Add testing for Python 3.11

7.2.4 1.0.0 (2022-12-16)

- Use SOFA conventions of version 2.1 from https://github.com/pyfar/sofa_conventions
- Verify SOFA data against all rules defined in the SOFA standard AES69-2022
- Add *Sofa.upgrade_convention* for upgrading outdated conventions. This now uses explicit upgrade rules from https://github.com/pyfar/sofa_conventions
- Remove upgrade functionality from *Sofa.verify*, *sofar.write_sofa*, and *sofar.read_sofa* for a more clear separation of functionality
- Add *Sofa.add_missing* to add missing default data to a SOFA object using the default values specified by the SOFA convention
- Add default parameter value to *Sofa.info*
- Make *sofar.update_conventions* a public function again
- Improve documentation and verbosity of command line output
- Add private function to check congruency of conventions stored as part of SOFAtoolbox and on sofaconventions.org
- Move to Circle CI and improve testing

7.2.5 0.3.1 (2022-03-21)

- Improvement *sofar.read*: Files with unknown Convention versions can now be read by updating to the latest or a specific version.
- Improvement *sofar.read*: Reporting custom variables when reading SOFA files from disk is now optional and no longer a warning.
- Improvement *Sofa.inspect*: SOFA objects that violate the SOFA convention can now be inspected. In this case, the violations are printed as message instead of raising an Error.
- Improvement *Sofa.verify*: SOFA objects can now be verified without any output in case the output is not desired when calling *Sofa.inspect*.

7.2.6 0.3.0 (2022_03_02)

- Feature: Add *sofar.inspect* function to get a quicker and better overview of the data inside a SOFA object
- Documentation: Add example of plotting HRIRs/HRTFs on the horizontal plane using pyfar>=0.4.0

7.2.7 0.2.0 (2022_02_14)

- Feature: Add *Sofa.delete* function to delete optional variables and attributes from SOFA objects
- Bugfix: sofar.read_sofa added data with default values from the SOFA convention even if the data were not contained in the SOFA-files. This is now fixed.
- Bugfix: N:LongName (attribute for SOFA conventions of Type TF, TF-E and TFE) is now optional as defined in AES69-2020.
- Improvement: Do not change time stamp of SOFA files in *sofar.read_sofa*
- Improvement: Multi-unit strings, e.g., 'degree, degree, meter' can now also be separated by spaces or commas only, e.g., 'degree degree,meter' as suggested by AES69-2020 (Issue #21)

- Improvement: Add testing for creating, writing, and reading Sofa files containing only mandatory data.

7.2.8 0.1.4 (2021-12-03)

- Bugfix: Patch for correctly creating Sofa objects if the path to sofar contains underscores '_'

7.2.9 0.1.3 (2021-11-19)

- Testing: Add missing dependency to setup.py
- Testing: Only test wheel during CI

7.2.10 0.1.2 (2021-11-18)

- Bugfix: Patch for correctly loading SOFA files with custom data

7.2.11 0.1.1 (2021-11-12)

- Documentation: Add examples for using pyfar to work with sofar and SOFA files

7.2.12 0.1.0 (2021-10-29)

- First release on PyPI

PYTHON MODULE INDEX

S

INDEX