EBU ADM Renderer (EAR)

Release 2.0.0

EBU ADM Renderer Authors

Nov 17, 2021

CONTENTS:

The **EBU ADM Renderer (EAR)** is a complete interpretation of the **Audio Definition Model (ADM)** format, specified in Recommendation ITU-R BS.2076-1. ADM is the recommended format for all stages and use cases within the scope of programme productions of **Next Generation Audio (NGA)**.

This documentation applies to the python reference implementation of the EAR, which can be found at https://github.com/ebu/ebu_adm_renderer.

This EAR is capable of rendering audio signals to all reproduction systems mentioned in "Advanced sound system for programme production (ITU-R BS.2051-1)".

Further descriptions of the EAR algorithms and functionalities can be found in EBU Tech 3388.

From version 2.0, this is also the reference implementation of ITU-R BS.2127 (ITU ADM Renderer).

CHAPTER

ONE

INSTALLATION

For best results, follow this Three Step Plan to installing the EAR without messing up your system python installation:

1) *Install Python*
2) *Use a Virtualenv*
3) *Install EAR*

1.1 Install Python

EAR requires Python version 3.6+. Recent python releases include virtualenv by default, so there's no need to install it separately.

Debian/Ubuntu `sudo apt install python3`

OSX `brew install python`

OSX includes python by default, but it's often outdated and configured a bit strangely, so it's best to install it from homebrew.

Windows https://www.python.org/downloads/windows/

It will probably work with tools like anaconda, pyenv, pipenv, poetry etc., but these are not necessary for most work, and are not actively tested.

1.2 Use a Virtualenv

A virtualenv (or *virtual environment*, or *venv*) is a self-contained python installation, containing the interpreter and a set of libraries and programs in a directory in your file system.

For information about how this should be used on different systems, refer to the official virtualenv guide.

In short, to create a virtualenv called `env` in the current directory:

```
python3 -m venv env
```

(you may have to adjust `python3` to the version which you installed above)

To activate it run:

```
source env/bin/activate
```

Now `pip` and `python` in this shell will operate within the virtualenv – pip will install packages into it, and python will only see packages installed into it. You'll have to activate the virtualenv in any other shell session which you want to work in.

If you want to use other python tools with the EAR (ipython, jupyter etc.) you should install and run them from the same virtualenv.

1.3 Install EAR

To install the latest published version:

```
pip install ear
```

Check that the install was successful by running `ear-render --help` – you should see the help message.

For development, or to try out the latest version, clone the repository and install it in *editable* mode instead:

```
git clone https://github.com/ebu/ebu_adm_renderer.git
cd ebu_adm_renderer
pip install -e .
```

Installed like this, any changes to the source will be visible without having to re-install.

You may want to install the extra tools required for testing and development at the same time:

```
pip install -e .[test,dev]
```

CHAPTER

TWO

USAGE

2.1 Command-Line Tools

The EAR reference implementation comes with two command line tools:

ear-render Is the main tool to render BW64/ADM audio files

ear-utils Collection of useful ADM utilities

2.1.1 ear-render

To render an ADM file, the following three parameters must be given:

- `-s` followed by the target output format to render to
- the name of the input file
- the name of the output file

For example, `ear-render -s 0+5+0 input.wav output_surround.wav` will render the BW64/ADM file `input.wav` to a `0+5+0` target speaker layout and store the result in `output_surround.wav`.

```
usage: ear-render [-h] [-d] -s target_system [-l layout_file]
                  [--output-gain-db gain_db] [--fail-on-overload]
                  [--enable-block-duration-fix] [--programme id]
                  [--comp-object id]
                  [--apply-conversion {to_cartesian,to_polar}] [--strict]
                  input_file output_file
```

Positional Arguments

input_file	BW64 file with CHNA and AXML (optional) chunks
output_file	BW64 output file

Named Arguments

-d, --debug print debug information when an error occurs

Default: False

-s, --system Target output system, accoring to ITU-R BS.2051. Available systems are: 0+2+0, 0+5+0, 2+5+0, 4+5+0, 4+5+1, 3+7+0, 4+9+0, 9+10+3, 0+7+0, 4+7+0

-l, --layout Layout config file

See *speakers_file*.

--output-gain-db output gain in dB (default: 0)

Default: 0

--fail-on-overload, -c fail if an overload condition is detected in the output

Default: False

--enable-block-duration-fix automatically try to fix faulty block format durations

Default: False

--programme select an audioProgramme to render by ID

--comp-object select an audioObject by ID from a complementary group

Default: []

--apply-conversion Possible choices: to_cartesian, to_polar

Apply conversion to Objects audioBlockFormats before rendering

--strict treat unknown ADM attributes as errors

Default: False

2.1.2 `ear-utils`

The `ear-utils` command contains a collection of utilities for working with ADM files as sub-commands.

EBU ADM renderer utilities

```
usage: ear-utils [-h] [-d]
                 {make_test_bwf,replace_axml,dump_axml,dump_chna,regenerate,ambix_to_bwf}
                 ...
```

Named Arguments

-d, --debug print debug information when an error occurs

Default: False

Sub-commands:

make_test_bwf

make a bwf file from a wav file and some metadata

```
ear-utils make_test_bwf [-h] -i INPUT -m META [--screen speakers_file] output
```

Positional Arguments

output	output bwf file

Named Arguments

-i, --input	input wav file
-m, --meta	input yaml metadata file
--screen	YAML format speakers file to take reference screen from See *speakers_file*.

replace_axml

replace the axml chunk in an existing ADM BWF file

```
ear-utils replace_axml [-h] -a file [-g] input output
```

Positional Arguments

input	input bwf file
output	output bwf file

Named Arguments

-a, --axml	new axml chunk file
-g, --gen-chna	generate the CHNA information from the track UIDs Default: False

dump_axml

dump the axml chunk of an ADM BWF file to stdout

```
ear-utils dump_axml [-h] input
```

Positional Arguments

input input bwf file

dump_chna

dump the chna chunk of an ADM BWF file to stdout

```
ear-utils dump_chna [-h] [-b] input
```

Positional Arguments

input input bwf file

Named Arguments

-b, --binary output binary data

Default: False

regenerate

read and write an ADM BWF file, regnerating the ADM and CHNA

```
ear-utils regenerate [-h] [--enable-block-duration-fix] input output
```

Positional Arguments

input input bwf file

output output bwf file

Named Arguments

--enable-block-duration-fix automatically try to fix faulty block format durations

Default: False

ambix_to_bwf

make a BWF file from an ambix format HOA file

```
ear-utils ambix_to_bwf [-h] [--norm NORM] [--nfcDist NFCDIST] [--screenRef]
                       [--chna-only]
                       input output
```

Positional Arguments

input	input file
output	output BWF file

Named Arguments

--norm	normalization mode
	Default: "SN3D"
--nfcDist	Near-Field Compensation Distance (float)
--screenRef	Screen Reference
	Default: False
--chna-only	use only CHNA with common definitions
	Default: False

2.2 Speakers File Format

Information about the loudspeaker layout can be passed to the renderer by using a speakers file with the `--speakers` flag.

2.2.1 File Format

A speakers file is a YAML document, which contains a list of loudspeakers under the `speakers` key, and the screen information under the `screen` key. Either may be omitted if not required.

Speakers list

The top level `speakers` item should contain a sequence of mappings, one for each output loudspeaker.

Each mapping should look something like this:

```
- {channel: 7, names: M+000, position: {az: 0.0, el: 0.0, r: 2.0 }}
```

which defines a loudspeaker connected to channel 7 (zero based), assigned to M+000 (in bs.2051 terms), with a given position. The file should contain a sequence of lines as above; one line per speaker.

The possible keys are as follows:

`channel` (required) The zero-based output channel number.

`names` (required) A list (or a single string) of BS.2051 channel names that this speaker should handle, i.e. like `M+000` or `[U+180, UH+180]`.

`position` (optional) A mapping containing the real loudspeaker position, with keys `az`, `el` and `r` specifying the azimuth, elevation and distance of the loudspeaker in ADM angle format (anticlockwise azimuth, degrees) and metres.

`gain_linear` (optional) A linear gain to apply to this output channel; this is useful for LFE outputs.

Screen

The top level `screen` item should contain a mapping, with at least a `type` key, and the following options, depending on the type. If the screen key is omitted, the default polar screen position specified in BS.2076-1 will be assumed. If a null screen is specified, then screen-related processing will not be applied.

if `type == "polar"`

`aspectRatio` (required) Screen width divided by screen height

`centrePosition` (required) Polar position of the centre of the screen, in the same format as the speaker `position` attribute.

`widthAzimuth` (required) Width of the screen in degrees.

if `type == "cart"`

`aspectRatio` (required) Screen width divided by screen height

`centrePosition` (required) Cartesian position of the centre of the screen; a mapping with keys `X`, `Y` and `Z`.

`widthX` (required) Width of the screen in Cartesian coordinates.

2.2.2 Examples

Useful speakers files should be stored in `ear/doc/speakers_files/`.

A minimal example with a polar screen would look like:

```
speakers:
    - {channel: 0, names: M+030, position: {az: 30.0, el: 0.0, r: 2.0 }}
    - {channel: 1, names: M-030, position: {az: -30.0, el: 0.0, r: 2.0 }}
screen:
    type: polar
    aspectRatio: 1.78
    centrePosition: {az: 0.0, el: 0.0, r: 1.0}
    widthAzimuth: 58.0
```

A minimal example with a Cartesian screen would look like:

```
speakers:
    - {channel: 0, names: M+030, position: {az: 30.0, el: 0.0, r: 2.0 }}
    - {channel: 1, names: M-030, position: {az: -30.0, el: 0.0, r: 2.0 }}
screen:
    type: cart
    aspectRatio: 1.78
    centrePosition: {X: 0.0, Y: 1.0, Z: 0.0}
    widthX: 0.5
```

A minimal example with screen processing disabled:

```
speakers:
    - {channel: 0, names: M+030, position: {az: 30.0, el: 0.0, r: 2.0 }}
    - {channel: 1, names: M-030, position: {az: -30.0, el: 0.0, r: 2.0 }}
screen: null
```

CHAPTER

THREE

DEVELOPER DOCUMENTATION

These pages contain documentation for people wanting to use the EAR python APIs to work with ADM data.

The EAR was primarily developed as a stand-alone application for rendering ADM files to loudspeaker signals, to support the standardisation of renderers. As such, the internal APIs are not necessarily as tidy and stable as they could be, as they were not the focus during development. You have been warned!

That being said, they represent one of the more complete packages for working with ADM data (along with libbw64 and libadm), so we welcome changes that improve the experience of using them.

3.1 File IO

The `ear.fileio` module contains functionality for reading and writing BW64 files, both with and without ADM content, a class structure for representing ADM metadata which can be parsed from and converted to XML, and various utilities for working with ADM data in this structure.

3.1.1 BW64 I/O

To read or write a BW64 file, the primary interface is the *openBw64Adm()* and *openBw64()* functions.

To read samples and ADM metadata (as an *ADM* object) from a file, use something like:

```
from ear.fileio import openBw64Adm

with openBw64Adm("path/to/file.wav") as f:
    adm = f.adm # get the ADM metadata

    for sample_block in f.iter_sample_blocks(1024):
        # do something with sample_block, which will be a numpy float array
        # of (nsamples, nchannels)
        print(sample_block.shape)
```

For lower level access without parsing ADM data:

```
from ear.fileio import openBw64

with openBw64("path/to/file.wav") as f:
    print(f.axml) # get the raw AXML data
    print(f.chna) # get the CHNA data

    while True:
```

(continues on next page)

(continued from previous page)

```
        sample_block = f.read(1024)
        if not len(sample_block):
            break
        print(sample_block.shape)
```

To write a file, you have to construct the format chunk manually:

```
from ear.fileio.bw64.chunks import FormatInfoChunk, ChnaChunk, AudioID
import numpy as np

# dummy ADM data
axml = b'some AXML data here'
chna = ChnaChunk([
    AudioID(1, 'ATU_00000001', 'AT_00010001_01', 'AP_00010003'),
])

formatInfo = FormatInfoChunk(formatTag=1,
                             channelCount=1,
                             sampleRate=48000,
                             bitsPerSample=24)

with openBw64("path/to/file.wav", "w", formatInfo=formatInfo) as f:
    # optionally write axml and chna data
    f.axml = axml
    f.chna = chna

    # write some sample blocks
    for i in range(10):
        f.write(np.zeros((1024, 1)))
```

To write some generated adm data, use something like this to generate the CHNA and AXML chunk data:

```
from ear.fileio.adm.chna import populate_chna_chunk
from ear.fileio.adm.generate_ids import generate_ids
from ear.fileio.adm.xml import adm_to_xml
import lxml.etree

adm = ...

generate_ids(adm)

chna = ChnaChunk()
populate_chna_chunk(chna, adm)

xml = adm_to_xml(adm)
axml = lxml.etree.tostring(xml, pretty_print=True)
```

See also:

generate_ids(), *populate_chna_chunk()*, *adm_to_xml()*. For generating ADM metadata, see *ADM Builder*.

These functions and classes are documented below:

ear.fileio.**openBw64**(*filename*, *mode='r'*, ***kwargs*)
Open a BW64 file for reading or writing.

Parameters

- **filename** (*str*) – file name
- **mode** (*str*) – `r` for read, or `w` for write
- **kwargs** – Extra arguments for Bw64Reader or Bw64Writer

Returns file object

Return type *Bw64Reader* or *Bw64Writer*

`ear.fileio.openBw64Adm(filename, fix_block_format_durations=False)`
Open a BW64 ADM file for reading. This automatically parses the ADM data, adds the common definitions, and adds information from the CHNA chunk. This can be accessed through the `.adm` attribute of the returned Bw64AdmReader.

Parameters

- **filename** (*str*) – file name
- **fix_block_format_durations** (*bool*) – attempt to fix rounding errors in audioBlockFormat durations

Returns file object

Return type *Bw64AdmReader*

`class ear.fileio.utils.Bw64AdmReader(bw64FileHandle, fix_block_format_durations=False)`
Utility for reading ADM data from a BW64 file; use *openBw64Adm()* to create these.

adm
ADM data

Type *ADM*

property bitdepth
number of bits per sample

property channels
number of channels

property chna
CHNA data

Type *bw64.chunks.ChnaChunk*

iter_sample_blocks(*blockSize*)
Read blocks of samples from the file.

Parameters **blockSize** (*int*) – number of samples to read at a time

Yields *np.ndarray of float* – sample blocks of shape (nsamples, nchannels), where nsamples is <= blockSize, and nchannels is the number of channels

property sampleRate
sample rate in Hz

property selected_items
default list of rendering items

Type list of ear.core.metadata_input.RenderingItem

`class ear.fileio.bw64.Bw64Reader(buffer)`
Read a WAVE/RF64/BW64 file.

Only PCM data (16bit, 24bit, 32bit) is currently supported. The class provides easy access to the axml, chna, bext chunks. The most important format information (samplerate, sample rate, bit rate, ...) can be directly accessed as properties.

property axml
data contained in axml chunk

Type bytes or None

property bext
data contained in bext chunk

Type bytes or None

property bitdepth
number of bits per sample

property channels
number of channels

property chna
CHNA data

Type *chunks.ChnaChunk* or None

get_chunk_data(*chunk_name*)
Read and return the binary data of a named chunk.

read(*numberOfFrames*)
read up to numberOfFrames samples

Returns sample blocks of shape (nsamples, nchannels), where nsamples is <= numberOfFrames, and nchannels is the number of channels

Return type np.ndarray of float

property sampleRate
sample rate in Hz

tell()
Get the sample number of the next sample returned by read.

class ear.fileio.bw64.**Bw64Writer**(*buffer, formatInfo=[formatTag: 1 channelCount: 1 sampleRate: 48000 bytesPerSecond: 96000 blockAlignment: 2 bitsPerSample: 16], chna=None, axml=None, bext=None, forceBw64=False*)

close()
Close and finalize the BW64 output.

This means that the final chunk sizes will be written to the buffer.

If you forget to call this function, the resulting file will be corrupted. Thus, it might be a good idea to use this with a contextmanager.

write(*samples*)
Append sample data to the BW64 data chunk.

Parameters **samples** (*array - like, dtype float*) – Array of audio samples, columns correspond to channels Expects float sample values in the range(-1, 1).

Chunk Classes

These classes represent chunks (or parts of chunks) in a BW64 file:

class ear.fileio.bw64.chunks.**ChnaChunk**(*audioIDs=NOTHING*)
Class representation of the ChannelAllocationChunk

audioIDs
CHNA entries

Type list of AudioID

asByteArray()
Get the binary representation of this chunk data.

class ear.fileio.bw64.chunks.**AudioID**(*trackIndex*, *audioTrackUID*, *audioTrackFormatIDRef*, *audioPackFormatIDRef*)
Class representation of a chna audioIDs list entry.

trackIndex
1-based index of the track in the sample data

Type int

audioTrackUID
audioTrackUID of the track

Type str

audioTrackFormatIDRef
audioTrackFormatID of the track

Type str

audioPackFormatIDRef
optional audioPackFormatID of the track

Type str or None

class ear.fileio.bw64.chunks.**FormatInfoChunk**(*formatTag=1*, *channelCount=1*, *sampleRate=48000*, *bytesPerSecond=None*, *blockAlignment=None*, *bitsPerSample=16*, *cbSize=None*, *extraData=None*)
Class representation of the FormatChunk

This class can be either used to create a new format chunk or simplify reading and validation of a format chunk. Once created the object cannot be changed. You can only read the saved data. The order of the constructor arguments might seem a bit strange at first sight. The order corresponds to the order within a BW64 file. This makes it easier to create an object from the data read from a file. Cumbersome values like bytesPerSecond or blockAlignment can be omitted (thus set to *None*). But: if they are set, they have to be correct. Otherwise a ValueError is raised.

To simplify the writing of files the FormatChunk (like every Chunk class in this module) has a asByteArray method. This method returns the correct byte array representation of the FormatChunk, which can be directly written to a file.

asByteArray()

property bitsPerSample

property blockAlignment

property bytesPerSecond

property cbSize

property channelCount

property extraData

property formatTag

property sampleRate

3.1.2 ADM Data

The ADM data representation is in the *ear.fileio.adm* module.

An ADM document is represented by an *adm.ADM* object, which contains lists of all of the top-level ADM elements.

In general, element objects have properties which match the ADM XML tag or attribute names, except for:

- The main ID of elements are stored in an `.id` property, rather than (for example) `.audioProgrammeID`.
- `typeDefinition` and `typeLabel` are resolved to a single `.type` property.
- `formatDefinition` and `formatLabel` are resolved to a single `.format` property.
- References to other objects by ID are translated into a python object reference, or a list of references. For example, `audioObjectIDRef` elements in an `audioContent` tag become a list of *elements.AudioObject* stored in `elements.AudioContent.audioObjects`.

  Note: Element objects do contain `IDRef` properties (e.g. `audioObjectIDRef`), which are used while parsing, but these are cleared when references are resolved to avoid storing conflicting information.

- In representations of ADM elements which contain both text and attributes (for example `<objectDivergence azimuthRange="30">0.5</objectDivergence>`), the text part is stored in a semantically-relevant property, e.g. *elements.ObjectDivergence.value*. For boolean elements (e.g. `channelLock`), this is represented by the presence or absence of the object in the parent object.

class ear.fileio.adm.adm.**ADM**

An ADM document.

addAudioChannelFormat(*channelformat*)

Add an audioChannelFormat.

Parameters **channelformat** (ear.fileio.adm.elements.AudioChannelFormat) –

addAudioContent(*content*)

Add an audioContent.

Parameters **content** (ear.fileio.adm.elements.AudioContent) –

addAudioObject(*audioobject*)

Add an audioObject.

Parameters **audioobject** (ear.fileio.adm.elements.AudioObject) –

addAudioPackFormat(*packformat*)

Add an audioPackFormat.

Parameters **packformat** (ear.fileio.adm.elements.AudioPackFormat) –

addAudioProgramme(*programme*)

Add an audioProgramme.

Parameters **programme** (ear.fileio.adm.elements.AudioProgramme) –

addAudioStreamFormat(*streamformat*)
Add an audioStreamFormat.

Parameters **streamformat** (ear.fileio.adm.elements.AudioStreamFormat) –

addAudioTrackFormat(*trackformat*)
Add an audioTrackFormat.

Parameters **trackformat** (ear.fileio.adm.elements.AudioTrackFormat) –

addAudioTrackUID(*trackUID*)
Add an audioTrackUID.

Parameters **trackUID** (ear.fileio.adm.elements.AudioTrackUID) –

property audioChannelFormats
Get all audioChannelFormat elements.

Type list[*AudioChannelFormat*]

property audioContents
Get all audioContent elements.

Type list[*AudioContent*]

property audioObjects
Get all audioObject elements.

Type list[*AudioObject*]

property audioPackFormats
Get all audioPackFormat elements.

Type list[*AudioPackFormat*]

property audioProgrammes
Get all audioProgramme elements.

Type list[*AudioProgramme*]

property audioStreamFormats
Get all audioStreamFormat elements.

Type list[*AudioStreamFormat*]

property audioTrackFormats
Get all audioTrackFormat elements.

Type list[*AudioTrackFormat*]

property audioTrackUIDs
Get all audioTrackUID elements.

Type list[*AudioTrackUID*]

property elements
Iterator over all elements.

lookup_element(*key*)
Get an element by ID.

validate()
Validate all elements, raising an exception if an error is found.

Note: This is not extensive.

Top-level Elements

class ear.fileio.adm.elements.main_elements.**ADMElement**(*id=None*, *is_common_definition=False*)

base class for top-level ADM elements

id

ADM ID attribute (e.g. audioProgrammeID)

Type str

is_common_definition

was this read from the common definitions file?

Type bool

class ear.fileio.adm.elements.**AudioProgramme**(*id=None*, *is_common_definition=False*, *audioProgrammeName=None*, *audioProgrammeLanguage=None*, *start=None*, *end=None*, *maxDuckingDepth=None*, *audioContents=NOTHING*, *audioContentIDRef=NOTHING*, *referenceScreen=PolarScreen(aspectRatio=1.78, centrePosition=PolarPosition(azimuth=0.0, elevation=0.0, distance=1.0), widthAzimuth=58.0)*, *loudnessMetadata=NOTHING*)

ADM audioProgramme

audioProgrammeName

Type str

audioProgrammeLanguage

Type Optional[str]

start

Type Optional[fractions.Fraction]

end

Type Optional[fractions.Fraction]

maxDuckingDepth

Type Optional[float]

audioContents

audioContent elements referenced via audioContentIDRef

Type list[*AudioContent*]

referenceScreen

Type Optional[Union[*CartesianScreen*, *PolarScreen*]]

loudnessMetadata

Type list[*LoudnessMetadata*]

class ear.fileio.adm.elements.**AudioContent**(*id=None*, *is_common_definition=False*, *audioContentName=None*, *audioContentLanguage=None*, *loudnessMetadata=NOTHING*, *dialogue=None*, *audioObjects=NOTHING*, *audioObjectIDRef=None*)

ADM audioContent

audioContentName

Type str

audioContentLanguage

Type Optional[str]

loudnessMetadata

Type list[*LoudnessMetadata*]

dialogue

Type Optional[int]

audioObject

Type list[*AudioObject*]

class ear.fileio.adm.elements.**AudioObject**(*id=None*, *is_common_definition=False*, *audioObjectName=None*, *start=None*, *duration=None*, *importance=None*, *interact=None*, *disableDucking=None*, *dialogue=None*, *audioPackFormats=NOTHING*, *audioTrackUIDs=NOTHING*, *audioObjects=NOTHING*, *audioComplementaryObjects=NOTHING*, *audioPackFormatIDRef=None*, *audioTrackUIDRef=None*, *audioObjectIDRef=None*, *audioComplementaryObjectIDRef=None*)

ADM audioObject

audioObjectName

Type str

start

Type Optional[fractions.Fraction]

duration

Type Optional[fractions.Fraction]

importance

Type Optional[int]

interact

Type Optional[bool]

disableDucking

Type Optional[bool]

dialogue

Type Optional[int]

audioPackFormats

Type list[*AudioPackFormat*]

audioTrackUIDs

Type list[*AudioTrackUID*]

audioObjects

Type list[*AudioObject*]

audioComplementaryObjects

Type list[*AudioObject*]

class ear.fileio.adm.elements.**AudioPackFormat**(*id=None*, *is_common_definition=False*, *audioPackFormatName=None*, *type=None*, *absoluteDistance=None*, *audioChannelFormats=NOTHING*, *audioPackFormats=NOTHING*, *importance=None*, *encodePackFormats=NOTHING*, *inputPackFormat=None*, *outputPackFormat=None*, *normalization=None*, *nfcRefDist=None*, *screenRef=None*, *audioChannelFormatIDRef=None*, *audioPackFormatIDRef=None*, *encodePackFormatIDRef=None*, *decodePackFormatIDRef=None*, *inputPackFormatIDRef=None*, *outputPackFormatIDRef=None*)

ADM audioPackformat

audioPackFormatName

Type str

type

`typeDefintion` and/or `typeLabel`

Type *TypeDefinition*

absoluteDistance

Type Optional[float]

audioChannelFormats

Type list[*AudioChannelFormat*]

audioPackFormats

Type list[*AudioPackFormat*]

importance

Type Optional[int]

encodePackFormats

Only for type==Matrix. Encode and decode pack references are a single binary many-many relationship; we only store one side.

Type list[*AudioPackFormat*]

inputPackFormat

Only for type==Matrix.

Type Optional[*AudioPackFormat*]

outputPackFormat
Only for type==Matrix.

Type Optional[*AudioPackFormat*]

normalization
Only for type==HOA.

Type Optional[str]

nfcRefDist
Only for type==HOA.

Type Optional[float]

screenRef
Only for type==HOA.

Type Optional[bool]

class ear.fileio.adm.elements.**AudioChannelFormat**(*id=None*, *is_common_definition=False*, *audioChannelFormatName=None*, *type=None*, *audioBlockFormats=NOTHING*, *frequency=NOTHING*)

ADM audioChannelFormat

audioChannelFormatName

Type str

type
`typeDefintion` and/or `typeLabel`

Type *TypeDefinition*

audioBlockFormats

Type list[*AudioBlockFormat*]

frequency

Type *Frequency*

class ear.fileio.adm.elements.**AudioTrackFormat**(*id=None*, *is_common_definition=False*, *audioTrackFormatName=None*, *format=None*, *audioStreamFormat=None*, *audioStreamFormatIDRef=None*)

ADM audioTrackFormat

audioTrackFormatName

Type str

format
`formatDefintion` and/or `formatLabel`

Type *FormatDefinition*

audioStreamFormat

Type Optional[*AudioStreamFormat*]

class ear.fileio.adm.elements.**AudioStreamFormat**(*id=None*, *is_common_definition=False*, *audioStreamFormatName=None*, *format=None*, *audioChannelFormat=None*, *audioPackFormat=None*, *audioTrackFormatIDRef=None*, *audioChannelFormatIDRef=None*, *audioPackFormatIDRef=None*)

ADM audioStreamFormat

audioStreamFormatName

Type str

format

formatDefintion and/or formatLabel

Type *FormatDefinition*

audioChannelFormat

Type Optional[*AudioStreamFormat*]

audioPackFormat

Type Optional[*AudioPackFormat*]

class ear.fileio.adm.elements.**AudioTrackUID**(*id=None*, *is_common_definition=False*, *trackIndex=None*, *sampleRate=None*, *bitDepth=None*, *audioTrackFormat=None*, *audioPackFormat=None*, *audioTrackFormatIDRef=None*, *audioPackFormatIDRef=None*)

ADM audioTrackUID

trackIndex

1-based track index from CHNA chunk

Type Optional[int]

sampleRate

Type Optional[int]

bitDepth

Type Optional[int]

audioTrackFormat

Type Optional[*AudioTrackFormat*]

audioPackFormat

Type Optional[*AudioPackFormat*]

Common Sub-Elements

class ear.fileio.adm.elements.**TypeDefinition**(*value*)
Bases: enum.Enum

ADM type definitions, representing `typeDefintion` and `typeLabel` attributes.

Binaural = 5

DirectSpeakers = 1

HOA = 4

Matrix = 2

Objects = 3

class ear.fileio.adm.elements.**FormatDefinition**(*value*)
Bases: enum.Enum

ADM format definitions, representing `formatDefintion` and `formatLabel` attributes.

PCM = 1

class ear.fileio.adm.elements.**LoudnessMetadata**(*loudnessMethod=None*, *loudnessRecType=None*, *loudnessCorrectionType=None*, *integratedLoudness=None*, *loudnessRange=None*, *maxTruePeak=None*, *maxMomentary=None*, *maxShortTerm=None*, *dialogueLoudness=None*)

ADM loudnessMetadata

loudnessMethod

Type Optional[str]

loudnessRecType

Type Optional[str]

loudnessCorrectionType

Type Optional[str]

integratedLoudness

Type Optional[float]

loudnessRange

Type Optional[float]

maxTruePeak

Type Optional[float]

maxMomentary

Type Optional[float]

maxShortTerm

Type Optional[float]

dialogueLoudness

Type Optional[float]

class ear.fileio.adm.elements.**Frequency**(*lowPass=None*, *highPass=None*)
ADM frequency element

lowPass

Type Optional[float]

highPass

Type Optional[float]

Common Types

class ear.fileio.adm.elements.**ScreenEdgeLock**(*horizontal=None*, *vertical=None*)
ADM screenEdgeLock information from position elements

horizontal
screenEdgeLock from azimuth or X coordinates; must be `left` or `right`.

Type Optional[str]

vertical
screenEdgeLock from elevation or Z coordinates; must be `top` or `bottom`.

Type Optional[str]

audioBlockFormat types

class ear.fileio.adm.elements.**AudioBlockFormat**(*id=None*, *rtime=None*, *duration=None*)
ADM audioBlockFormat base class

id

Type Optional[str]

rtime

Type Optional[fractions.Fraction]

duration

Type Optional[fractions.Fraction]

Objects audioBlockFormat

class ear.fileio.adm.elements.**AudioBlockFormatObjects**(*id=None*, *rtime=None*, *duration=None*, *position=None*, *cartesian=False*, *width=0.0*, *height=0.0*, *depth=0.0*, *gain=1.0*, *diffuse=0.0*, *channelLock=None*, *objectDivergence=None*, *jumpPosition=NOTHING*, *screenRef=False*, *importance=10*, *zoneExclusion=NOTHING*)
Bases: *ear.fileio.adm.elements.AudioBlockFormat*

ADM audioBlockFormat with typeDefinition == "Objects"

position

Type Optional[*ObjectPosition*]

cartesian

Type bool

width

Type float

height

Type float

depth

Type float

gain

Type float

diffuse

Type float

channelLock

Type Optional[*ChannelLock*]

objectDivergence

Type Optional[*ObjectDivergence*]

jumpPosition

Type *JumpPosition*

screenRef

Type bool

importance

Type int

zoneExclusion

Type list[Union[*CartesianZone*, *PolarZone*]]

class ear.fileio.adm.elements.**CartesianZone**(*minX*, *minY*, *minZ*, *maxX*, *maxY*, *maxZ*)

ADM zoneExclusion zone element with Cartesian coordinates

minX

Type float

minY

Type float

minZ

Type float

maxX

Type float

maxY

Type float

maxZ

Type float

class ear.fileio.adm.elements.**PolarZone**(*minElevation*, *maxElevation*, *minAzimuth*, *maxAzimuth*)
ADM zoneExclusion zone element with polar coordinates

minElevation

Type float

maxElevation

Type float

minAzimuth

Type float

maxAzimuth

Type float

class ear.fileio.adm.elements.**ChannelLock**(*maxDistance=None*)
ADM channelLock element

maxDistance

Type Optional[float]

class ear.fileio.adm.elements.**JumpPosition**(*flag=False*, *interpolationLength=None*)
ADM jumpPosition element

flag
contents of the jumpPosition element

Type bool

interpolationLength

Type Optional[fractions.Fraction]

class ear.fileio.adm.elements.**ObjectDivergence**(*value*, *azimuthRange=None*, *positionRange=None*)
ADM objectDivergence element

value

Type float

azimuthRange

Type Optional[float]

positionRange

Type Optional[float]

class ear.fileio.adm.elements.**ObjectPosition**
Base for classes representing data contained in *audioBlockFormat position* elements for Objects.

See also:

ObjectPolarPosition and *ObjectCartesianPosition*

class ear.fileio.adm.elements.**ObjectPolarPosition**(*azimuth*, *elevation*, *distance=1.0*, *screenEdgeLock=NOTHING*)
Bases: *ear.fileio.adm.elements.ObjectPosition*, *ear.common.PolarPositionMixin*

Represents data contained in *audioBlockFormat position* elements for Objects where polar coordinates are used.

Attributes are formatted according to the ADM coordinate convention.

azimuth
anti-clockwise azimuth in degrees, measured from the front

Type float

elevation
elevation in degrees, measured upwards from the equator

Type float

distance
distance relative to the audioPackFormat absoluteDistance parameter

Type float

screenEdgeLock

Type *ScreenEdgeLock*

class ear.fileio.adm.elements.**ObjectCartesianPosition**(*X*, *Y*, *Z*, *screenEdgeLock=NOTHING*)
Bases: *ear.fileio.adm.elements.ObjectPosition*, *ear.common.CartesianPositionMixin*

Represents data contained in *audioBlockFormat position* elements for Objects where Cartesian coordinates are used.

Attributes are formatted according to the ADM coordinate convention.

X
left-to-right position, from -1 to 1

Type float

Y
back-to-front position, from -1 to 1

Type float

Z
bottom-to-top position, from -1 to 1

Type float

screenEdgeLock

Type *ScreenEdgeLock*

DirectSpeakers audioBlockFormat

class ear.fileio.adm.elements.**AudioBlockFormatDirectSpeakers**(*id=None*, *rtime=None*, *duration=None*, *position=None*, *speakerLabel=NOTHING*)
Bases: *ear.fileio.adm.elements.AudioBlockFormat*

ADM audioBlockFormat with typeDefinition == "DirectSpeakers"

position

Type *DirectSpeakerPosition*

speakerLabel

Type list[str]

class ear.fileio.adm.elements.**BoundCoordinate**(*value*, *min=None*, *max=None*)
ADM position coordinate for DirectSpeakers

This represents multiple position elements with the same coordinate, so for azimuth this translates to:

```
<position coordinate="azimuth">{value}</position>
<position coordinate="azimuth" bound="min">{min}</position>
<position coordinate="azimuth" bound="max">{max}</position>
```

Attributes are formatted according to the ADM coordinate convention.

value
value for unbounded position element

Type float

min
value for position element with `bound="min"`

Type Optional[float]

max
value for position element with `bound="max"`

Type Optional[float]

class ear.fileio.adm.elements.**DirectSpeakerPosition**
Base for classes representing data contained in *audioBlockFormat position* elements for DirectSpeakers.

See also:

DirectSpeakerPolarPosition and *DirectSpeakerCartesianPosition*

class ear.fileio.adm.elements.**DirectSpeakerPolarPosition**(*bounded_azimuth*, *bounded_elevation*, *bounded_distance=NOTHING*, *screenEdgeLock=NOTHING*)
Bases: *ear.fileio.adm.elements.DirectSpeakerPosition*, *ear.common.PolarPositionMixin*

Represents data contained in *audioBlockFormat position* elements for DirectSpeakers where polar coordinates are used.

Attributes are formatted according to the ADM coordinate convention.

bounded_azimuth
data for position elements with `coordinate="azimuth"`

Type *BoundCoordinate*

bounded_elevation
data for position elements with `coordinate="elevation"`

Type *BoundCoordinate*

bounded_distance
data for position elements with `coordinate="distance"`

Type *BoundCoordinate*

screenEdgeLock

Type *ScreenEdgeLock*

as_cartesian_array()
Get the position as a Cartesian array.

Returns equivalent X, Y and Z coordinates

Return type np.array of shape (3,)

property azimuth
anti-clockwise azimuth in degrees, measured from the front

Type float

property distance
distance relative to the audioPackFormat absoluteDistance parameter

Type float

property elevation
elevation in degrees, measured upwards from the equator

Type float

class ear.fileio.adm.elements.**DirectSpeakerCartesianPosition**(*bounded_X*, *bounded_Y*, *bounded_Z*, *screenEdgeLock=NOTHING*)

Bases: *ear.fileio.adm.elements.DirectSpeakerPosition*, *ear.common.CartesianPositionMixin*

Represents data contained in *audioBlockFormat position* elements for DirectSpeakers where Cartesian coordinates are used.

bounded_X
data for position elements with `coordinate="X"`

Type *BoundCoordinate*

bounded_Y
data for position elements with `coordinate="Y"`

Type *BoundCoordinate*

bounded_Z
data for position elements with `coordinate="Z"`

Type *BoundCoordinate*

screenEdgeLock

Type *ScreenEdgeLock*

property X
left-to-right position, from -1 to 1

Type float

property Y
back-to-front position, from -1 to 1

Type float

property Z
bottom-to-top position, from -1 to 1

Type float

HOA AudioBlockFormat

class ear.fileio.adm.elements.**AudioBlockFormatHoa**(*id=None*, *rtime=None*, *duration=None*, *equation=None*, *order=None*, *degree=None*, *normalization=None*, *nfcRefDist=None*, *screenRef=None*)

Bases: *ear.fileio.adm.elements.AudioBlockFormat*

ADM audioBlockFormat with typeDefinition == "HOA"

equation

Type Optional[str]

order

Type Optional[int]

degree

Type Optional[int]

normalization

Type Optional[str]

nfcRefDist

Type Optional[float]

screenRef

Type Optional[bool]

Matrix AudioBlockFormat

class ear.fileio.adm.elements.**AudioBlockFormatMatrix**(*id=None*, *rtime=None*, *duration=None*, *outputChannelFormat=None*, *matrix=NOTHING*, *outputChannelFormatIDRef=None*)

Bases: *ear.fileio.adm.elements.AudioBlockFormat*

ADM audioBlockFormat with typeDefinition == "Matrix"

outputChannelFormat

Type Optional[*AudioChannelFormat*]

matrix

Type list[*MatrixCoefficient*]

class ear.fileio.adm.elements.**MatrixCoefficient**(*inputChannelFormat=None*, *gain=None*, *gainVar=None*, *phase=None*, *phaseVar=None*, *delay=None*, *delayVar=None*, *inputChannelFormatIDRef=None*)

ADM audioBlockFormat Matrix coefficient element

inputChannelFormat

Type Optional[*AudioChannelFormat*]

gain

Type Optional[float]

gainVar

Type Optional[str]

phase

Type Optional[float]

phaseVar

Type Optional[str]

delay

Type Optional[float]

delayVar

Type Optional[str]

3.1.3 ADM XML Handling

XML Parsing

ear.fileio.adm.xml.**load_axml_doc**(*adm*, *element*, *lookup_references=True*, *fix_block_format_durations=False*)

Load some axml into an ADM structure.

This is a low-level function and doesn't deal with common definitions.

Parameters

- **adm** (ADM) – ADM structure to add to
- **element** (*lxml.etree._Element*) – parsed ADM XML
- **lookup_references** (*bool*) – should we look up references?
- **fix_block_format_durations** (*bool*) – should we attempt to fix up inaccuracies in audioBlockFormat durations?

Note: This is deprecated; use the functions in *ear.fileio.adm.timing_fixes* instead.

ear.fileio.adm.xml.**load_axml_string**(*adm*, *axmlstr*, ***kwargs*)

Wrapper around *load_axml_doc()* which parses XML too.

Parameters

- **adm** (ADM) – ADM structure to add to
- **axmlstr** (*str*) – ADM XML string
- **kwargs** – see *load_axml_doc()*

ear.fileio.adm.xml.**load_axml_file**(*adm*, *axmlfile*, ***kwargs*)

Wrapper around *load_axml_doc()* which loads XML from a file.

Parameters

- **adm** (ADM) – ADM structure to add to

- **axmlfile** (*Union[str, File]*) – ADM XML file name or file object; see lxml.etree.parse().
- **kwargs** – see *load_axml_doc()*

ear.fileio.adm.xml.**parse_string**(*axmlstr*, ***kwargs*)
Parse an ADM XML string, including loading common definitions.

Parameters

- **axmlstr** (*str*) – ADM XML string
- **kwargs** – see *load_axml_doc()*

Returns ADM structure

Return type *ADM*

ear.fileio.adm.xml.**parse_file**(*axmlfile*, ***kwargs*)
Parse an ADM XML file, including loading common definitions.

Parameters

- **axmlfile** (*Union[str, File]*) – ADM XML file name or file object; see lxml.etree.parse().
- **kwargs** – see *load_axml_doc()*

Returns ADM structure

Return type *ADM*

XML Generation

ear.fileio.adm.xml.**adm_to_xml**(*adm*)
Generate an XML element corresponding to an ADM structure.

This skips elements marked with is_common_definition

Parameters **adm** (ear.fileio.adm.adm.ADM) –

Return type lxml.etree._Element

3.1.4 ADM Utilities

ADM Builder

The *builder.ADMBuilder* class makes it easier to construct basic ADM structures which are properly linked together. For example, to make an ADM with a single Objects channel:

```
from ear.fileio.adm.builder import ADMBuilder
from ear.fileio.adm.elements import AudioBlockFormatObjects, ObjectPolarPosition

builder = ADMBuilder()

builder.create_programme(audioProgrammeName="my audioProgramme")
builder.create_content(audioContentName="my audioContent")

block_formats = [
    AudioBlockFormatObjects(
```

(continues on next page)

(continued from previous page)

```
        position=ObjectPolarPosition(azimuth=0.0, elevation=0.0, distance=1.0),
    ),
]
builder.create_item_objects(0, "MyObject 1", block_formats=block_formats)

# do something with builder.adm
```

class ear.fileio.adm.builder.**ADMBuilder**(*adm=NOTHING*, *last_programme=None*, *last_content=None*, *last_object=None*, *last_pack_format=None*, *last_stream_format=None*, *item_parent=None*)

Builder object for creating ADM object graphs.

adm
ADM object to modify.

Type *ADM*

last_programme
The last programme created, to which created audioContents will be linked.

Type Optional[*AudioProgramme*]

last_content
The last content created, to which created audioObjects will be linked by default.

Type Optional[*AudioContent*]

last_object
The last object created, to which created audioObjects or audioPackFormats will be linked by default.

Type Optional[*AudioObject*]

last_pack_format
The last pack_format created, to which created audioChannelFormats will be linked by default.

Type Optional[*AudioPackFormat*]

last_stream_format
The last stream_format created, to which created audioTrackFormats will be linked by default.

Type Optional[*AudioStreamFormat*]

item_parent
The last explicitly created audioContent or audioObject, used as the parent for audioObjects created by create_item* functions.

Type Optional[Union[*AudioContent*, *AudioObject*]]

class Format(*channel_formats*, *track_formats*, *pack_format*, *stream_formats*)
Structure referencing the ADM components of a format with a particular channel layout.

This holds an audioPackFormat, and one audioTrackFormat, audioStreamFormat and audioChannelFormat per channel in the format.

channel_formats
Type list[*AudioChannelFormat*]

track_formats
Type list[*AudioTrackFormat*]

pack_format
Type *AudioPackFormat*

stream_formats
Type list[*AudioStreamFormat*]

property channel_format
singular accessor for channel_formats

property stream_format
singular accessor for stream_formats

property track_format
singular accessor for track_formats

class Item(*format*, *track_uids*, *audio_object*, *parent*)
Structure referencing the ADM components of a created item.

This holds an audioObject, one audioTrackUID per channel, and a reference to `Format` for the format parts of the item.

format
Type *Format*

track_uids
Type list[*AudioTrackUID*]

audio_object
Type *AudioObject*

parent
Type Union[*AudioContent*, *AudioObject*]

property channel_format
accessor for format.channel_format

property channel_formats
accessor for format.channel_formats

property pack_format
accessor for format.pack_format

property stream_format
accessor for format.stream_format

property stream_formats
accessor for format.stream_formats

property track_format
accessor for format.track_format

property track_formats
accessor for format.track_formats

property track_uid
singular accessor for track_uids

MonoItem
compatibility alias for users expecting *_mono to return MonoItem

alias of *ear.fileio.adm.builder.ADMBuilder.Item*

create_channel(*parent=DEFAULT*, ***kwargs*)
Create a new audioChannelFormat.

Parameters

- **parent** (`AudioPackFormat`) – parent packFormat; defaults to the last packFormat created

- **kwargs** – see *AudioChannelFormat*

Returns created audioChannelFormat

Return type *AudioChannelFormat*

create_content(*parent=DEFAULT*, ***kwargs*)
Create a new audioContent.

Parameters

- **parent** (AudioProgramme) – parent programme; defaults to the last one created
- **kwargs** – see *AudioContent*

Returns created audioContent

Return type *AudioContent*

create_format_hoa(*orders*, *degrees*, *name*, ***kwargs*)
Create ADM components representing the format of a HOA stream.

This is a wrapper around *create_format_multichannel()*.

Parameters

- **orders** (*list[int]*) – order for each track
- **degrees** (*list[int]*) – degree for each track
- **name** (*str*) – name used for all components
- **kwargs** – arguments for *AudioBlockFormatHoa*

Returns the created components

Return type *Format*

create_format_mono(*type*, *name*, *block_formats=[]*)
Create ADM components needed to represent a mono format.

This makes:

- an audioChannelFormat with the given block_formats
- an audioPackFormat linked to the audioChannelFormat
- an audioStreamFormat linked to the audioChannelFormat
- an audioTrackFormat linked to the audioStreamFormat

Parameters

- **type** (TypeDefinition) – type of channelFormat and packFormat
- **name** (*str*) – name used for all components
- **block_formats** (*list[AudioBlockFormat]*) – block formats to add to the channel format

Returns the created components

Return type *Format*

create_format_multichannel(*type*, *name*, *block_formats*)
Create ADM components representing a multi-channel format.

This makes:

- an audioChannelFormat for each channel
- an audioStreamFormat linked to each audioChannelFormat
- an audioTrackFormat linked to each audioStreamFormat
- an audioPackFormat linked to the audioChannelFormats

Parameters

- **type** (TypeDefinition) – type of channelFormat and packFormat
- **name** (*str*) – name used for all components
- **block_formats** (*list[list[AudioBlockFormat]]*) – list of audioBlockFormats for each audioChannelFormat

Returns the created components

Return type *Format*

create_item_direct_speakers(**args*, ***kwargs*)

Create ADM components needed to represent a DirectSpeakers channel.

Wraps *create_item_mono()* with `type=TypeDefinition.DirectSpeakers`.

Returns the created components

Return type *Item*

create_item_hoa(*track_indices*, *orders*, *degrees*, *name*, *parent=DEFAULT*, ***kwargs*)

Create ADM components representing a HOA stream.

This is a wrapper around *create_format_hoa()* and *create_item_multichannel_from_format()*.

Parameters

- **track_indices** (*list[int]*) – zero-based indices of the tracks in the BWF file.
- **orders** (*list[int]*) – order for each track
- **degrees** (*list[int]*) – degree for each track
- **name** (*str*) – name used for all components
- **parent** (*Union[AudioContent, AudioObject]*) – parent of the created audioObject defaults to the last content or explicitly created object
- **kwargs** – arguments for *AudioBlockFormatHoa*

Returns the created components

Return type *Item*

create_item_mono(*type*, *track_index*, *name*, *parent=DEFAULT*, *block_formats=[]*)

Create ADM components needed to represent a mono channel, either DirectSpeakers or Objects.

This makes:

- an audioChannelFormat with the given block_formats
- an audioPackFormat linked to the audioChannelFormat
- an audioStreamFormat linked to the audioChannelFormat
- an audioTrackFormat linked to the audioStreamFormat
- an audioTrackUID linked to the audioTrackFormat and audioPackFormat

- an audioObject linked to the audioTrackUID and audioPackFormat

Parameters

- **type** (TypeDefinition) – type of channelFormat and packFormat
- **track_index** (*int*) – zero-based index of the track in the BWF file.
- **name** (*str*) – name used for all components
- **parent** (*Union[AudioContent, AudioObject]*) – parent of the created audioObject defaults to the last content or explicitly created object
- **block_formats** (*list[AudioBlockFormat]*) – block formats to add to the channel format

Returns the created components

Return type *Item*

create_item_mono_from_format(*format*, *track_index*, *name*, *parent=DEFAULT*)

Create ADM components needed to represent a mono channel given an existing format.

This makes:

- an audioTrackUID linked to the audioTrackFormat and audioPackFormat of format
- an audioObject linked to the audioTrackUID and audioPackFormat

Parameters

- **format** (Format) –
- **track_index** (*int*) – zero-based index of the track in the BWF file.
- **name** (*str*) – name used for all components
- **parent** (*Union[AudioContent, AudioObject]*) – parent of the created audioObject defaults to the last content or explicitly created object

Returns the created components

Return type *Item*

create_item_multichannel(*type*, *track_indices*, *name*, *block_formats*, *parent=DEFAULT*)

Create ADM components representing a multi-channel object.

This makes:

- an audioChannelFormat for each channel
- an audioStreamFormat linked to each audioChannelFormat
- an audioTrackFormat linked to each audioStreamFormat
- an audioPackFormat linked to the audioChannelFormats
- an audioTrackUID linked to each audioTrackFormat and the audioPackFormat
- an audioObject linked to the audioTrackUIDs and the audioPackFormat

Parameters

- **type** (TypeDefinition) – type of channelFormat and packFormat
- **track_indices** (*list[int]*) – zero-based indices of the tracks in the BWF file.

- **name** (*str*) – name used for all components
- **block_formats** (*list[list[AudioBlockFormat]]*) – list of audioBlockFormats for each audioChannelFormat
- **parent** (*Union[AudioContent, AudioObject]*) – parent of the created audioObject defaults to the last content or explicitly created object

Returns the created components

Return type *Item*

create_item_multichannel_from_format(*format*, *track_indices*, *name*, *parent=DEFAULT*)

Create ADM components representing a multi-channel object, referencing an existing format structure.

This makes:

- an audioTrackUID linked to each audioTrackFormat and the audioPackFormat in format
- an audioObject linked to the audioTrackUIDs and the audioPackFormat in format

Parameters

- **format** (*Format*) – format components to reference
- **track_indices** (*list[int]*) – zero-based indices of the tracks in the BWF file.
- **name** (*str*) – name used for all components
- **parent** (*Union[AudioContent, AudioObject]*) – parent of the created audioObject defaults to the last content or explicitly created object

Returns the created components

Return type *Item*

create_item_objects(**args*, ***kwargs*)

Create ADM components needed to represent an object channel.

Wraps *create_item_mono()* with `type=TypeDefinition.Objects`.

Returns the created components

Return type *Item*

create_object(*parent=DEFAULT*, ***kwargs*)

Create a new audioObject.

Parameters

- **parent** (*Union[AudioContent, AudioObject]*) – parent content or object; defaults to the last content created
- **kwargs** – see *AudioObject*

Returns created audioObject

Return type *AudioObject*

create_pack(*parent=DEFAULT*, ***kwargs*)

Create a new audioPackFormat.

Parameters

- **parent** (*AudioObject or AudioPackFormat*) – parent object or packFormat; defaults to the last object created

- **kwargs** – see *AudioPackFormat*

Returns created audioPackFormat

Return type *AudioPackFormat*

create_programme(***kwargs*)
Create a new audioProgramme.

Parameters **kwargs** – see *AudioProgramme*

Returns created audioProgramme

Return type *AudioProgramme*

create_stream(***kwargs*)
Create a new audioStreamFormat.

Parameters **kwargs** – see AudioChannelFormat

Returns created audioStreamFormat

Return type *AudioStreamFormat*

create_track(*parent=DEFAULT*, ***kwargs*)
Create a new audioTrackFormat.

Parameters

- **parent** (AudioStreamFormat) – parent streamFormat; defaults to the last audioStreamFormat created
- **kwargs** – see AudioTrackFormat

Returns created audioTrackFormat

Return type *AudioTrackFormat*

create_track_uid(*parent=DEFAULT*, ***kwargs*)
Create a new audioTrackUID.

Parameters

- **parent** (AudioObject) – parent audioObject; defaults to the last audioObject created
- **kwargs** – see AudioTrackUID

Returns created audioTrackUID

Return type *AudioTrackUID*

load_common_definitions()
Load common definitions into adm.

ID Generation

When ADM objects are created, they have their IDs set to `None`. Before serialisation, the IDs must be generated using `generate_ids()`:

ear.fileio.adm.generate_ids.**generate_ids**(*adm*)
regenerate ids for all elements in adm

Parameters **adm** (ADM) – ADM structure to modify

CHNA Utilities

In a BW64 file, the AXML and CHNA chunks store overlapping and related information about audioTrackUIDs:

- track index: CHNA only
- audioTrackFormat reference: CHNA and AXML
- audioPackFormat reference: CHNA and AXML

To simplify this, the *elements.AudioTrackUID* class stores the track index from the CHNA, and we provide utilities for copying data between a CHNA and an ADM object:

ear.fileio.adm.chna.**load_chna_chunk**(*adm*, *chna*)
Add information from the chna chunk to the adm track UIDs structure.

Any existing track UIDs in adm are checked for consistency against the data in chna; any non-existent track-UIDs are created.

The track index, pack format ref and track format ref attributes are transferred; there references are resolved, and we assume that any references in adm have already been resolved

Parameters

- **adm** (ADM) – adm structure to add information to
- **chna** (ChnaChunk) – chna chunk to get information from

ear.fileio.adm.chna.**populate_chna_chunk**(*chna*, *adm*)
Populate the CHNA chunk with information from the ADM model.

All CHNA entries are replaced, and are populated from the trackIndex, audioTrackUID, and the track format/pack format references.

Since the CHNA chunk contains IDs, the IDs in the ADM must have been generated before calling this.

Parameters

- **adm** (ADM) – adm structure to get information to
- **chna** (ChnaChunk) – chna chunk to populate

ear.fileio.adm.chna.**guess_track_indices**(*adm*)
Guess track indices from the audioTrackUID IDs.

This information should really come from the CHNA, but sometimes that isn't available.

Parameters **adm** (ADM) – ADM structure to modify

Common Definitions

The library includes a copy of the common definitions file, which can be loaded into an ADM structure:

ear.fileio.adm.common_definitions.**load_common_definitions**(*adm*)
Load the common definitions file from IRU-R Rec. BS.2094-1, setting is_common_definition=True on all loaded elements.

Parameters **adm** (ADM) – ADM structure to add to.

3.1.5 ADM Exceptions

Various ADM-related exceptions are defined:

class ear.fileio.adm.exceptions.**AdmError**
Bases: Exception

Base class for ADM parsing exceptions.

class ear.fileio.adm.exceptions.**AdmMissingRequiredElement**
Bases: *ear.fileio.adm.exceptions.AdmError*

Exception raised for missing required elements.

class ear.fileio.adm.exceptions.**AdmIDError**
Bases: *ear.fileio.adm.exceptions.AdmError*

Exception raised when errors relating to IDs are identified.

class ear.fileio.adm.exceptions.**AdmFormatRefError**(*message*, *reasons*)
Bases: *ear.fileio.adm.exceptions.AdmError*

Error in references between ADM content and format parts.

3.1.6 Timing Fixes

This module contains functions which can be used to fix common timing issues in ADM files. Generally these are caused by rounding of *start*, *rtime* and *duration* parameters.

The following timing issues are addressed:

- *audioBlockFormats* where the *rtime* plus the *duration* of one *audioBlockFormat* does not match the *rtime* of the next.
- *interpolationTime* parameter larger than *duration*.
- *audioBlockFormat rtime* plus *duration* extending past the end of the containing *audioObject*.

ear.fileio.adm.timing_fixes.**check_blockFormat_timings**(*adm*, *fix=False*)
If fix, fix various audioBlockFormat timing issues, otherwise just issue warnings.

Parameters **adm** (ear.fileio.adm.adm.ADM) – ADM document to modify

ear.fileio.adm.timing_fixes.**fix_blockFormat_timings**(*adm*)
Fix various audioBlockFormat timing issues, issuing warnings.

Parameters **adm** (ear.fileio.adm.adm.ADM) – ADM document to modify

3.2 Core

The ear.core module contains functionality for rendering ADM data and audio to loudspeaker signals, and various components supporting that functionality. In general, an implementation of everything written in Recommendation ITU-R BS.2127-0 can be found somewhere in this module.

3.2.1 Metadata Input

The data structures in the `ear.core.metadata_input` module act as the main interface between the core renderer classes for each type, and the wider EAR system, or other systems which want to render audio using the EAR.

The data structures are *not ADM*: they are essentially a simplified view of ADM data containing only the parts required for rendering: an association between some input audio tracks, and streams of time-varying metadata to apply to them.

The input to a renderer is a list of *RenderingItem* objects, which are specialised for each ADM type (*ObjectRenderingItem*, *DirectSpeakersRenderingItem* etc.). Each rendering item contains:

- A pointer to some audio data, through a *TrackSpec* object. This generally says something like "track 2 of the input audio stream", but can also contain a more complex mapping for Matrix types, or reference a silent input.

  In the case of HOA where multiple tracks are rendered together, multiple track specs are given.

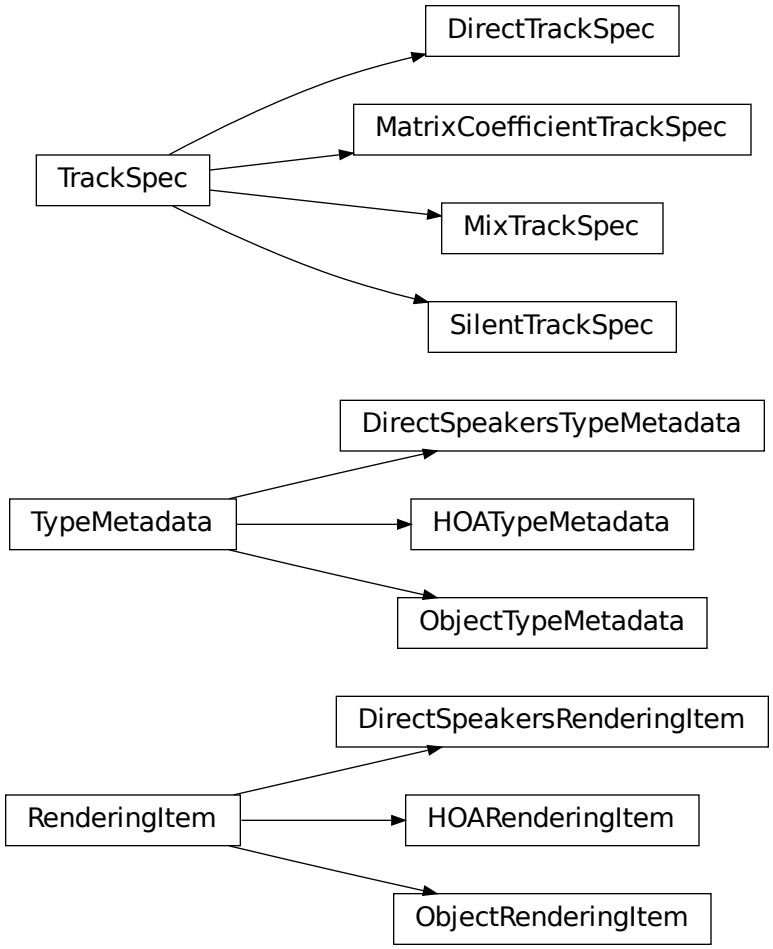
  See *Track Specs*.

- A source of time-varying metadata, provided by a *MetadataSource*. This can be used to access a sequence of *TypeMetadata* objects, which are again sub-classed for each ADM type (*ObjectTypeMetadata*, *DirectSpeakersTypeMetadata* etc.).

  TypeMetadata sub-classes generally contain a pointer to the corresponding audioBlockFormat data, as well as any extra data from outside the audioBlockFormat which is needed for rendering.

- Extra data (generally *ImportanceData* and *ADMPath*) which is not required for rendering, but may be useful for debugging, or uses other than straightforward rendering (for example only rendering some sub-set of an ADM file).

The available classes and their inheritance relationships are shown below:



Overall Structure

class ear.core.metadata_input.**RenderingItem**
Base class for *RenderingItem classes; these should represent an item to be rendered, combining a MetadataSource that produces a sequence of TypeMetadata objects, and some indices into the tracks that this metadata applies to.

class ear.core.metadata_input.**TypeMetadata**
Base class for *TypeMetadata classes; these should represent all the parameters needed to render some set of audio channels within some time bounds.

class ear.core.metadata_input.**MetadataSource**
A source of metadata for some input channels.

get_next_block()
Get the next metadata block, if one is available.

Return type Optional[*ear.core.metadata_input.TypeMetadata*]

Track Specs

To render track specs, see *ear.core.track_processor.TrackProcessor()* and *ear.core.track_processor.MultiTrackProcessor*.

class ear.core.metadata_input.**TrackSpec**
Represents a method for obtaining audio samples to be processed given multi-track input samples (from a WAV file for example).

This is used to abstract over regular track references, silent channels and matrix channels, but could also be used for coded audio, fancy containers etc.

class ear.core.metadata_input.**DirectTrackSpec**(*track_index*)
Bases: *ear.core.metadata_input.TrackSpec*

Track obtained directly from the input audio stream.

track_index
Zero based input track index.

Type int

class ear.core.metadata_input.**SilentTrackSpec**
Bases: *ear.core.metadata_input.TrackSpec*

A track whose samples are always 0.

class ear.core.metadata_input.**MatrixCoefficientTrackSpec**(*input_track*, *coefficient*)
Bases: *ear.core.metadata_input.TrackSpec*

Track derived from a single channel and a matrix coefficient.

input_track
track spec to obtain samples from before they are processed by parameters in coefficient.

Type *TrackSpec*

coefficient
matrix parameters to apply; inputChannelFormat should be ignored.

Type *MatrixCoefficient*

class ear.core.metadata_input.**MixTrackSpec**(*input_tracks*)
Bases: *ear.core.metadata_input.TrackSpec*

Track that is a mix of some other tracks.

input_tracks
list of input tracks to mix

Type list of TrackSpec

Objects

class ear.core.metadata_input.**ObjectTypeMetadata**(*block_format*, *extra_data=NOTHING*)
Bases: *ear.core.metadata_input.TypeMetadata*

TypeMetadata for typeDefinition="Objects"

block_format
Block format.

Type *AudioBlockFormatObjects*

extra_data
Extra parameters from outside block format.

Type *ExtraData*

class ear.core.metadata_input.**ObjectRenderingItem**(*track_spec*, *metadata_source*, *importance=NOTHING*, *adm_path=None*)

Bases: *ear.core.metadata_input.RenderingItem*

RenderingItem for typeDefinition="Objects"

track_spec
Zero based input track index for this item.

Type *TrackSpec*

metadata_source
Source of ObjectTypeMetadata objects.

Type *MetadataSource*

importance
Importance data for this item.

Type *ImportanceData*

adm_path
Pointers to the ADM objects which this is derived from.

Type *ADMPath*

Direct Speakers

class ear.core.metadata_input.**DirectSpeakersTypeMetadata**(*block_format*, *audioPackFormats=None*, *extra_data=NOTHING*)

Bases: *ear.core.metadata_input.TypeMetadata*

TypeMetadata for typeDefinition="DirectSpeakers"

block_format
Block format.

Type *AudioBlockFormatDirectSpeakers*

extra_data
Extra parameters from outside block format.

Type *ExtraData*

class ear.core.metadata_input.**DirectSpeakersRenderingItem**(*track_spec*, *metadata_source*, *importance=NOTHING*, *adm_path=None*)

Bases: *ear.core.metadata_input.RenderingItem*

RenderingItem for typeDefinition="DirectSpeakers"

track_spec
Specification of input samples.

Type *TrackSpec*

metadata_source
Source of DirectSpeakersTypeMetadata objects.

Type *MetadataSource*

importance
Importance data for this item.

Type *ImportanceData*

adm_path
Pointers to the ADM objects which this is derived from.

Type *ADMPath*

HOA

class ear.core.metadata_input.**HOATypeMetadata**(*orders*, *degrees*, *normalization*, *nfcRefDist=None*, *screenRef=False*, *rtime=None*, *duration=None*, *extra_data=NOTHING*)

Bases: *ear.core.metadata_input.TypeMetadata*

TypeMetadata for typeDefinition="HOA"

orders
Order for each input channel.

Type list of int

degrees
Degree for each channel.

Type list of int

normalization
Normalization for all channels.

Type str

nfcRefDist
NFC Reference distance for all channels.

Type float or None

screenRef
Are these channels screen related?

Type bool

rtime
Start time of block.

Type fractions.Fraction or None

duration
Duration of block.

Type fractions.Fraction or None

extra_data
Info from object and channels for all channels.

Type *ExtraData*

class ear.core.metadata_input.**HOARenderingItem**(*track_specs*, *metadata_source*, *importances=None*, *adm_paths=None*)

Bases: *ear.core.metadata_input.RenderingItem*

RenderingItem for typeDefinition="HOA"

track_specs
Specification of n tracks of input samples.

Type list[*TrackSpec*]

metadata_source
Source of HOATypeMetadata objects; will usually contain only one object.

Type *MetadataSource*

importances
Importance data for each track.

Type Optional[list[*ImportanceData*]]

adm_paths
Pointers to the ADM objects which each track is derived from.

Type Optional[list[*ADMPath*]]

Shared Data

class ear.core.metadata_input.**ExtraData**(*object_start=None*, *object_duration=None*, *reference_screen=PolarScreen(aspectRatio=1.78, centrePosition=PolarPosition(azimuth=0.0, elevation=0.0, distance=1.0), widthAzimuth=58.0)*, *channel_frequency=NOTHING*, *pack_absoluteDistance=None*)
Common metadata from outside the ADM block format.

object_start
Start time of audioObject.

Type fractions.Fraction or None

object_duration
Duration of audioObject.

Type fractions.Fraction or None

reference_screen
Reference screen from audioProgramme.

Type *CartesianScreen* or *PolarScreen*

channel_frequency
Frequency information from audioChannelFormat.

Type *Frequency*

pack_absoluteDistance
absoluteDistance parameter from audioPackFormat.

Type float or None

class ear.core.metadata_input.**ADMPath**(*audioProgramme=None*, *audioContent=None*, *audioObjects=None*, *audioPackFormats=None*, *audioChannelFormat=None*)
Pointers to the ADM objects which a rendering item is derived from.

audioProgramme

Type Optional[*AudioProgramme*]

audioContent

Type Optional[*AudioContent*]

audioObjects

Type Optional[list[*AudioObject*]]

audioPackFormats

Type Optional[list[*AudioPackFormat*]]

audioChannelFormat

Type Optional[list[*AudioChannelFormat*]]

property first_audioObject
The first audioObject of this track in the chain, or None

Type Optional[*AudioObject*]

property first_audioPackFormat
The first audioPackFormat of this track in the chain, or None

Type Optional[*AudioPackFormat*]

property last_audioObject
The last audioObject of this track in the chain, or None

Type Optional[*AudioObject*]

property last_audioPackFormat
The last audioPackFormat of this track in the chain, or None

Type Optional[*AudioPackFormat*]

class ear.core.metadata_input.**ImportanceData**(*audio_object=None*, *audio_pack_format=None*)
Importance metadata for a single channel in a RenderingItem

audio_object
Importance that has been derived from the audioObject level

Type int or None

audio_pack_format
Importance that has been derived from the audioPackFormat level

Type int or None

3.2.2 Rendering Item Selection

Rendering item selection is the process of turning an ADM document into a set of rendering items, as described in *Metadata Input*.

ear.core.select_items.**select_rendering_items**(*adm*, *audio_programme=None*, *selected_complementary_objects=[]*)
Select RenderingItems from an ADM structure.

Parameters

- **adm** (ADM) – ADM to process
- **audio_programme** (*Optional[AudioProgramme]*) – audioProgramme to select if there is more than one in adm.

- **selected_complementary_objects** (*list[AudioObject]*) – Objects to select from each complementary group. If there is no entry for an object in a complementary object group, then the root is selected by default.

Returns selected rendering items

Return type list[*RenderingItem*]

3.2.3 Track Processor

The `ear.core.track_processor` module can be used to render *Track Specs*. Users should create a *TrackProcessorBase* via *TrackProcessor()*, which can be used to extract a single channel of audio from a multi-channel bus.

MultiTrackProcessor allows for processing multiple tracks at once.

class ear.core.track_processor.**TrackProcessorBase**(*track_spec*)

Base class for processors which can be used to obtain samples for a single track spec given multi-track input samples (from a WAV file for example).

Use *TrackProcessor()* to create these.

process(*sample_rate*, *input_samples*)

Get the samples for one track spec.

Parameters

- **sample_rate** (*int*) – sample rate of input/output samples in Hz
- **input_samples** (*array of (n, c) floats*) – c channels of n input samples

Returns n samples for track_spec.

Return type array of (n,) floats

ear.core.track_processor.**TrackProcessor**(*track_spec*)

Build a processor to render a single track spec.

Parameters **track_spec** (TrackSpec) – Track spec to render.

Returns processor to obtain samples for track_spec

Return type *TrackProcessorBase*

class ear.core.track_processor.**MultiTrackProcessor**(*track_specs*)

A processor that renders multiple track specs into a single array given multi-track input samples (from a WAV file for example).

process(*sample_rate*, *input_samples*)

Get the samples for all track specs.

Parameters

- **sample_rate** (*int*) – sample rate of input/output samples in Hz
- **input_samples** (*array of (n, c) floats*) – c channels of n input samples

Returns n samples for each of the m track_specs.

Return type array of (n, m) floats

3.2.4 Loudspeaker Layout

The `ear.core.layout` module contains data structures which represent loudspeaker layouts. Rather than being constructed directly, these should be created by calling `ear.core.bs2051.get_layout()`, and modified to match real-world layouts using the functionality described in *Real-world Layouts*.

class ear.core.layout.**Layout**(*name*, *channels*, *screen=PolarScreen(aspectRatio=1.78, centrePosition=PolarPosition(azimuth=0.0, elevation=0.0, distance=1.0), widthAzimuth=58.0)*)

Representation of a loudspeaker layout, with a name and a list of channels.

name
layout name

Type str

channels
list of channels in the layout

Type list[*Channel*]

screen
screen information to use for screen-related content

Type Optional[Union[*CartesianScreen*, *PolarScreen*]]

property channel_names
The channel names for each channel.

Type list[str]

property channels_by_name
dict from channel name to Channel.

check_positions(*callback=<function _print_warning>*)
Call callback with error messages for any channel positions that are out of range.

check_upmix_matrix(*upmix*, *callback=<function _print_warning>*)
Call callback with error messages for any errors in an upmix matrix.

- each input channel should be routed to 1 output channel
- each output channel should be routed from 0 or 1 input channels

property is_lfe
Bool array corresponding to channels that selects LFE channels.

property nominal_positions
Nominal channel positions as an (n, 3) numpy array.

property norm_positions
Normalised channel positions as an (n, 3) numpy array.

property positions
Channel positions as an (n, 3) numpy array.

with_real_layout(*real_layout*)
Incorporate information from a real layout.

Note: see with_speakers for information on speaker mapping

Parameters **real_layout** (`RealLayout`) – real layout information to incorporate

Returns

- A new Layout object with updated speaker positions and screen information.
- An upmix matrix to map the loudspeakers to the correct channels and apply gains.

with_speakers(*speakers*)
Remap speaker positions to those in speakers, and produce an upmix matrix to map from the channels in the layout to the channels in the speaker list.

Parameters **speakers** (*list[Speaker]*) – list of speakers to map to.

Returns

- A new Layout object with the same channels but with positions matching those in speakers.
- An upmix matrix m, such that m.dot(x) will map values corresponding to the channels in self.channels to the channel numbers in speakers. This matrix may be missing entries or have duplicate entries depending on the contents of speakers; use check_upmix_matrix.

property without_lfe
The same layout, without LFE channels.

Type *Layout*

class ear.core.layout.**Channel**(*name*, *polar_position*, *polar_nominal_position=NOTHING*, *az_range=NOTHING*, *el_range=NOTHING*, *is_lfe=False*)
Representation of a channel, with a name, real and nominal positions, allowed azimuth and elevation ranges, and an lfe flag.

name
Channel name.

Type str

polar_position
real speaker location

Type *PolarPosition*, or arguments for PolarPosition

polar_nominal_position
nominal speaker location, defaults to polar_position

Type *PolarPosition*, or arguments for PolarPosition

az_range
azimuth range in degrees; allowed range is interpreted as starting at az_range[0], moving anticlockwise to az_range[1]; defaults to the azimuth of polar_nominal_position.

Type 2-tuple of floats

el_range
elevation range in degrees; allowed range is interpreted as starting at el_range[0], moving up to el_range[1]; defaults to the elevation of polar_nominal_position.

Type 2-tuple of floats

is_lfe
is this an LFE channel?

Type bool

check_position(*callback=<function _print_warning>*)
Call callback with an error message if the position is outside the azimuth and elevation ranges.

BS.2051 Layouts

ear.core.bs2051.**get_layout**(*name*)
Get data for a layout specified in BS.2051.

Parameters **name** (*str*) – Full layout name, e.g. "4+5+0"

Returns object representing the layout; real speaker positions are set to the nominal positions.

Return type *Layout*

Real-world Layouts

The following functionality is used to allow a user to specify the real position of loudspeakers in a listening room (independent of the layouts that can be created with them), which can be used to modify *layout.Layout* objects using *Layout.with_real_layout()*:

class ear.core.layout.**RealLayout**(*speakers=None*, *screen=PolarScreen(aspectRatio=1.78, centrePosition=PolarPosition(azimuth=0.0, elevation=0.0, distance=1.0), widthAzimuth=58.0)*)
Representation of a complete listening environment, onto which a standard layout will be mapped.

speakers
all speakers that could be used

Type Optional[list[*Speaker*]]

screen
screen information to use for screen-related content

Type Optional[Union[*CartesianScreen*, *PolarScreen*]]

class ear.core.layout.**Speaker**(*channel*, *names*, *polar_position=None*, *gain_linear=1.0*)
Representation of a real-world loudspeaker; an array of these represents the data required to use the renderer in a given listening room.

channel
0-based channel number

Type int

names
list of BS.2051 channel names this speaker should handle.

Type list[str]

polar_position
real loudspeaker position, if known

Type Optional[*PolarPosition*]

gain_linear
linear gain to apply to this output channel

Type float

ear.core.layout.**load_real_layout**(*fileobj*)
Load a real layout from a yaml file.

The format is either a list of objects representing speakers, or an object with optional keys `speakers` (which contains a list of objects representing speakers) and `screen` (which contains an object representing the screen).

Objects representing speakers may have the following keys:

channel 0-based channel number, required

names list (or a single string) of BS.2051 channel names that this speaker should handle, i.e. like `"M+000"` or `["U+180", "UH+180"]`

position optional associative array containing the real loudspeaker position, with keys:

az anti-clockwise azimuth in degrees

el elevation in degrees

r radius in metres

gain_linear optional linear gain to be applied to this channel

A polar screen may be represented with the following keys:

type `"polar"`, required

aspectRatio aspect ratio of the screen

centrePosition object representing the centre position of the screen:

az anti-clockwise azimuth in degrees

el elevation in degrees

r radius in metres

widthAzimuth width of the screen in degrees

A Cartesian screen may be represented with the following keys:

type `"cart"`, required

aspectRatio aspect ratio of the screen

centrePosition object representing the centre position of the screen containing X, Y and Z coordinates

widthX width of the screen along the Cartesian X axis

If the screen is omitted, the default screen is used; if the screen is specified but null, then screen-related processing will not be applied.

Parameters **`fileobj`** – a file-like object to read yaml from

Returns real layout information

Return type *RealLayout*

See *Speakers File Format* for more details and examples.

3.2.5 Conversion

The `ear.core.objectbased.conversion` module contains functionality to convert *`AudioBlockFormatObjects`* objects between polar and Cartesian coordinate conventions, according to section 10 of BS.2127-0.

Conversion functions in this module are not straightforward coordinate conversions, they instead try to minimise the difference in behaviour between the polar and Cartesian rendering paths.

Conversion can not account for all differences between the two rendering paths, and while conversion of position coordinates is invertible, conversion of extent parameters is not, except in simple cases. Because of these limitations, conversion should be used as part of production processes, where the results can be monitored and adjusted.

audioBlockFormat Conversion Functions

These conversions apply conversion to *AudioBlockFormatObjects* objects, returning a new copy:

ear.core.objectbased.conversion.**to_polar**(*block_format*)
Convert a block format to use polar coordinates according to ITU-R BS.2127-0 section 10.

The cartesian flag will be set to match the coordinates used.

The position, width, height and depth will be converted; the rest of the parameters will be unmodified.

Parameters **block_format** (ear.fileio.adm.elements.AudioBlockFormatObjects) –

Return type *ear.fileio.adm.elements.AudioBlockFormatObjects*

ear.core.objectbased.conversion.**to_cartesian**(*block_format*)
Convert a block format to use Cartesian coordinates according to ITU-R BS.2127-0 section 10.

The cartesian flag will be set to match the coordinates used.

The position, width, height and depth will be converted; the rest of the parameters will be unmodified.

Parameters **block_format** (ear.fileio.adm.elements.AudioBlockFormatObjects) –

Return type *ear.fileio.adm.elements.AudioBlockFormatObjects*

Low-Level Conversion Functions

These functions operate on the individual parameters, and may be useful for testing:

ear.core.objectbased.conversion.**point_polar_to_cart**(*az*, *el*, *d*)
Convert a position from polar to Cartesian according to ITU-R BS.2127-0 section 10.

Parameters

- **az** (*float*) – azimuth
- **el** (*float*) – elevation
- **d** (*float*) – distance

Returns converted Cartesian position

Return type np.ndarray of shape (3,)

ear.core.objectbased.conversion.**point_cart_to_polar**(*x*, *y*, *z*)
Convert a position from Cartesian to polar according to ITU-R BS.2127-0 section 10.

Parameters

- **x** (*float*) – X coordinate
- **y** (*float*) – Y coordinate
- **z** (*float*) – Z coordinate

Returns converted azimuth, elevation and distance

Return type (float, float, float)

ear.core.objectbased.conversion.**extent_polar_to_cart**(*az*, *el*, *dist*, *width*, *height*, *depth*)
Convert a position and extent parameters from polar to Cartesian according to ITU-R BS.2127-0 section 10.

Parameters

- **az** (*float*) – azimuth

- **el** (*float*) – elevation
- **dist** (*float*) – distance
- **width** (*float*) – width parameter
- **height** (*float*) – height parameter
- **depth** (*float*) – depth parameter

Returns converted X, Y, Z, width (X size), depth (Y size) and height (Z size)

Return type (float, float, float, float, float, float)

ear.core.objectbased.conversion.**extent_cart_to_polar**(*x*, *y*, *z*, *xs*, *ys*, *zs*)

Convert a position and extent parameters from Cartesian to polar according to ITU-R BS.2127-0 section 10.

Parameters

- **x** (*float*) – X coordinate
- **y** (*float*) – Y coordinate
- **z** (*float*) – Z coordinate
- **xs** (*float*) – width (X size)
- **ys** (*float*) – depth (Y size)
- **zs** (*float*) – height (Z size)

Returns converted azimuth, elevation, distance, width, height and depth

Return type (float, float, float, float, float, float)

3.2.6 Geometry Utilities

Coordinate Conversion

ear.core.geom.**cart**(*az*, *el*, *dist*, *axis=- 1*)

Convert ADM-format polar positions to ADM-format Cartesian.

Parameters

- **az** – Azimuths in degrees, angle measured anticlockwise from front.
- **el** – Elevations in degrees, angle measured up from equator.
- **r** – Radii.
- **axis** – Index of the new axis in the result; see `numpy.stack()`. -1 (default) adds a new axis at the end.

Returns Same shape as broadcasting az, el and r together, with a new axis at *axis* containing the X, Y and Z coordinates.

Return type ndarray

Examples

```
>>> cart(0, 0, 1)
array([0., 1., 0.])
>>> cart(90, 0, 1).round(6)
array([-1.,  0.,  0.])
>>> cart(0, 90, 1).round(6)
array([0., 0., 1.])
>>> # inputs are broadcast together...
>>> cart([0, 90], 0, 1).round(6)
array([[ 0.,  1.,  0.],
       [-1.,  0.,  0.]])
>>> # ... along the given axis
>>> cart([0, 90], 0, 1, axis=0).round(6)
array([[ 0., -1.],
       [ 1.,  0.],
       [ 0.,  0.]])
```

ear.core.geom.**azimuth**(*positions*, *axis=- 1*)

Get the azimuth in degrees from ADM-format Cartesian positions.

Parameters

- **positions** (*array of float*) – Cartesian positions, with X, Y and Z positions along axis *axis*.
- **axis** (*int*) – Axis to find coordinates along. -1 (default) indicates the last axis.

Returns

Azimuths of the positions in degrees; has the same shape as positions, with *axis* removed.

Return type array

Raises ValueError – If positions does not have the right length along axis.

Examples

```
>>> azimuth([0, 1, 0]).round(0).astype(int)
0
>>> azimuth([[1, 0, 0], [0, 1, 0]]).round(0).astype(int)
array([-90,   0])
>>> azimuth([[1, 0], [0, 1], [0, 0]], axis=0).round(0).astype(int)
array([-90,   0])
```

ear.core.geom.**elevation**(*positions*, *axis=- 1*)

Get the elevation in degrees from ADM-format Cartesian positions.

See *azimuth()*.

ear.core.geom.**distance**(*positions*, *axis=- 1*)

Get the distance from ADM-format Cartesian positions.

See *azimuth()*.

Angle Utilities

ear.core.geom.**relative_angle**(*x*, *y*)
Assuming y is clockwise from x, increment y by 360 until it's not less than x.

Parameters

- **x** (*float*) – start angle in degrees.
- **y** (*float*) – end angle in degrees.

Returns y shifted such that it represents the same angle but is greater than x.

Return type float

ear.core.geom.**inside_angle_range**(*x*, *start*, *end*, *tol=0.0*)
Assuming end is clockwise from start, is the angle x inside [start,end] within some tolerance?

Parameters

- **x** (*float*) – angle to test, in degrees
- **start** (*float*) – start angle of range, in degrees
- **end** (*float*) – end angle of range, in degrees
- **tol** (*float*) – tolerance in degrees to check within

Returns bool

3.3 Common Structures

3.3.1 Position Classes

class ear.common.**Position**
A 3D position represented in polar or Cartesian coordinates.

class ear.common.**PolarPosition**(*azimuth*, *elevation*, *distance=1.0*)
Bases: *ear.common.Position*, *ear.common.PolarPositionMixin*

A 3D position represented in ADM-format polar coordinates.

azimuth
anti-clockwise azimuth in degrees, measured from the front

Type float

elevation
elevation in degrees, measured upwards from the equator

Type float

distance
distance relative to the audioPackFormat absoluteDistance parameter

Type float

class ear.common.**CartesianPosition**(*X*, *Y*, *Z*)
Bases: *ear.common.Position*, *ear.common.CartesianPositionMixin*

A 3D position represented in ADM-format Cartesian coordinates.

X
left-to-right position, from -1 to 1

Type float

Y
back-to-front position, from -1 to 1

Type float

Z
bottom-to-top position, from -1 to 1

Type float

Position Mixins

These two classes provide common methods for the various real position classes:

class ear.common.**PolarPositionMixin**
Methods to be defined on all polar position objects which have azimuth, elevation and distance attributes.

as_cartesian_array()
Get the position as a Cartesian array.

Returns equivalent X, Y and Z coordinates

Return type np.array of shape (3,)

as_cartesian_position()
Get the equivalent cartesian position.

Return type *ear.common.CartesianPosition*

class ear.common.**CartesianPositionMixin**
Methods to be defined on all Cartesian position objects which have X, Y and Z attributes.

as_cartesian_array()
Get the position as a Cartesian array.

Returns equivalent X, Y and Z coordinates

Return type np.array of shape (3,)

as_polar_position()
Get the equivalent cartesian position.

Return type *ear.common.PolarPosition*

3.3.2 Screen Classes

class ear.common.**CartesianScreen**(*aspectRatio*, *centrePosition*, *widthX*)
ADM screen representation using Cartesian coordinates.

This is used to represent the audioProgrammeReferenceScreen, as well as the screen position in the reproduction room.

aspectRatio
aspect ratio

Type float

centrePosition
screenCentrePosition element

Type *CartesianPosition*

widthX
screenWidth X attribute

Type float

class ear.common.**PolarScreen**(*aspectRatio*, *centrePosition*, *widthAzimuth*)
ADM screen representation using Cartesian coordinates.

This is used to represent the audioProgrammeReferenceScreen, as well as the screen position in the reproduction room.

aspectRatio
aspect ratio

Type float

centrePosition
screenCentrePosition element

Type *PolarPosition*

widthX
screenWidth azimuth attribute

Type float

ear.common.**default_screen = PolarScreen(aspectRatio=1.78, centrePosition=PolarPosition(azimuth=0.0, elevation=0.0, distance=1.0), widthAzimuth=58.0)**
The default screen position, size and shape.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

V

W

X

Y

Z