jams Documentation

Release 0.0.1

Eric J. Humphrey, Justin Salamon, Oriol Nieto, Jon Forsyth, Rach

May 28, 2015

Contents

Contents:

CHAPTER 1

PyJAMS

Top-level module for pyjams. JAMS Python API

This library provides an interface for reading JAMS into Python, or creating them programatically.

1.1 1. Creating a JAMS data structure from scratch

First, create the top-level JAMS container:

```
>>> import pyjams
>>> jam = pyjams.JAMS()
```

Now we can create a beat annotation:

```
>>> annot = jam.beat.create_annotation()
>>> beat = annot.create_datapoint()
>>> beat.time.value = 0.33
>>> beat.time.confidence = 1.0
>>> beat.label.value = "1"
>>> beat.label.confidence = 0.75
```

Then, we'll update the annotation's metadata by directly setting its fields:

```
>>> annot.annotation_metadata.data_source = "Poorly paid students"
>>> annot.annotation_metadata.curator.name = "My Name"
>>> annot.annotation_metadata.curator.email = "somebody@aol.com"
```

And now a second time, cause this is our house (and we can do what we want):

```
>>> beat2 = annot.create_datapoint()
>>> beat2.label.value = "second beat"
```

Once you've added all your data, you can serialize the annotation to a file with the built-in *json* library:

```
>>> import json
>>> with open("these_are_my.jams", 'w') as fp:
        json.dump(annot, fp, indent=2)
```

Or, less verbosely, using the built-in save function:

```
>>> pyjams.save(annot, "these_are_still_my.jams")
```

1.2 2. Reading a Jams file

Assuming you already have a JAMS file on-disk, say at 'these_are_also_my.jams', you can easily read it back into memory:

```
>>> another_annot = pyjams.load('these_are_also_my.jams')
```

And that's it!

```
>>> print annot2
```

pyjams.pyjams.**load**(*filepath*)
Load a JAMS Annotation from a file.

pyjams.pyjams.**save**(*jam*, *filepath*)
Serialize annotation as a JSON formatted stream to file.

pyjams.pyjams.**append**(*jam*, *filepath*, *new_filepath=None*, *on_conflict='fail'*)
Append the contents of one JAMS file to another.

jam: JAMS object Annotation object to write.

filepath: str Jams file the object should be added to.

new_filepath: str Optional output file for non-destructive append operations.

on_conflict: str, default='fail'

Strategy for resolving metadata conflicts; one of: ['fail', 'overwrite', 'ignore'].

class pyjams.pyjams.**JObject**(***kwargs*)
Bases: object

Dict-like object for JSON Serialization.

This object behaves like a dictionary to allow init-level attribute names, seamless JSON-serialization, and double-star style unpacking (**obj).

keys()
Return the fields of the object.

update(***kwargs*)

type

class pyjams.pyjams.**Sandbox**(*unconstrained*)
Bases: *pyjams.pyjams.JObject*

Functionally identitical to JObjects, but the class hierarchy might be confusing if all objects inherit from Sandboxes.

class pyjams.pyjams.**Observation**(*global*)
Bases: *pyjams.pyjams.JObject*

Smallest observable concept (value) with a confidence interval. Used for almost anything, from observed times to semantic tags.

class pyjams.pyjams.**Event**(*sparse*, *instantaneous*)
Bases: *pyjams.pyjams.Observation*

Instantaneous time event, consisting of two Observations (time and label). Used for such ideas like beats or onsets.

class pyjams.pyjams.**Range**(*start=None*, *end=None*, *label=None*)
Bases: *pyjams.pyjams.Observation*

An observed time interval, composed of three Observations (start, end, and label). Used for such concepts as chords.

duration

class pyjams.pyjams.**TimeSeries**(*value=None*, *time=None*, *confidence=None*)
Bases: *pyjams.pyjams.Observation*

Sampled Time Series Observation

This could be an array, and skip the value abstraction. However, some abstraction could help turn data into numpy arrays on the fly.

However, np.ndarrays are not directly serializable. It might be necessary to subclass np.ndarray and change __repr__.

class pyjams.pyjams.**BaseAnnotation**(*data=None*, *annotation_metadata=None*, *sandbox=None*)
Bases: *pyjams.pyjams.JObject*

Annotation base class.

Default Type: None

Be aware that Annotations define a '_DefaultType' class variable, specifying the kind of objects contained in its 'data' attribute. Therefore any subclass will need to set this accordingly.

create_datapoint()
Factory method to create an empty Data object based on this type of Annotation, adding it to the data list and returning a reference.

obj: self._DefaultType An empty object, whose type is determined by the Annotation type.

class pyjams.pyjams.**ObservationAnnotation**(*data=None*, *annotation_metadata=None*, *sandbox=None*)
Bases: *pyjams.pyjams.BaseAnnotation*

Observation Annotation

Default Type: Observation

Be aware that Annotations define a '_DefaultType' class variable, specifying the kind of objects contained in its 'data' attribute. Therefore any subclass will need to set this accordingly.

class pyjams.pyjams.**EventAnnotation**(*data=None*, *annotation_metadata=None*, *sandbox=None*)
Bases: *pyjams.pyjams.BaseAnnotation*

Event Annotation

Default Type: Event

Be aware that Annotations define a '_DefaultType' class variable, specifying the kind of objects contained in its 'data' attribute. Therefore any subclass will need to set this accordingly.

labels
All labels in the annotation, as a single object.

labels: JObject Object with the label fields (value, confidence, secondary_label) as lists, in order over the annotation.

times
All times in the annotation, as a single object.

times: JObject Object with the time fields (value, confidence, secondary_label) as lists, in order over the annotation.

class pyjams.pyjams.**TimeSeriesAnnotation**(*data=None*, *annotation_metadata=None*, *sandbox=None*)

Bases: *pyjams.pyjams.BaseAnnotation*

TimeSeries Annotation

Default Type: TimeSeries

Be aware that Annotations define a '_DefaultType' class variable, specifying the kind of objects contained in its 'data' attribute. Therefore any subclass will need to set this accordingly.

class pyjams.pyjams.**RangeAnnotation**(*data=None*, *annotation_metadata=None*, *sandbox=None*)

Bases: *pyjams.pyjams.BaseAnnotation*

Range Annotation

Default Type: Range

Be aware that Annotations define a '_DefaultType' class variable, specifying the kind of objects contained in its 'data' attribute. Therefore any subclass will need to set this accordingly.

labels

All labels in the annotation, as a single object.

labels: JObject Object with the label fields (value, confidence, secondary_label) as lists, in order over the annotation.

starts

All start times in the annotation, as a single object.

start_times: JObject Object with the start fields (value, confidence, secondary_label) as lists, in order over the annotation.

ends

All end times in the annotation, as a single Observation.

end_times: Observation Object with the end fields (value, confidence, secondary_label) as lists, in order over the annotation.

intervals

All start and end times in the annotation.

intervals: list of tuples Ordered collection of (start.value, end.value) pairs

class pyjams.pyjams.**Curator**(*name=''*, *email=''*)

Bases: *pyjams.pyjams.JObject*

Container object for curator metadata.

class pyjams.pyjams.**AnnotationMetadata**(*curator=None*, *version=''*, *corpus=''*, *annotator=None*, *annotation_tools=''*, *annotation_rules=''*, *validation=''*, *data_source=''*)

Bases: *pyjams.pyjams.JObject*

Data structure for metadata corresponding to a specific annotation.

class pyjams.pyjams.**FileMetadata**(*title=''*, *artist=''*, *release=''*, *duration=''*, *identifiers=None*, *jams_version=None*)

Bases: *pyjams.pyjams.JObject*

Metadata for a given audio file.

class pyjams.pyjams.**AnnotationList**(*annotations*, *AnnotationType*)
Bases: list

List subclass for managing collections of annotations, providing factory methods to create empty annotations.

create_annotation()
Create an empty XAnnotation based on the annotation type provided on init, adding it to the annotation list and returning a reference to the new annotation object.

obj: AnnotationType An empty annotation, whose type is determined by self._DefaultType.

class pyjams.pyjams.**JAMS**(*beat=None*, *chord=None*, *genre=None*, *key=None*, *mood=None*, *melody=None*, *note=None*, *onset=None*, *pattern=None*, *pitch=None*, *segment=None*, *source=None*, *tag=None*, *file_metadata=None*, *sandbox=None*)
Bases: *pyjams.pyjams.JObject*

Top-level Jams Object

add(*jam*, *on_conflict='fail'*)
Add the contents of another jam to this object.

Note that, by default, this method fails if file_metadata is not identical and raises a ValueError; either resolve this manually (because conflicts should almost never happen), force an 'overwrite', or tell the method to 'ignore' the metadata of the object being added.

jam: JAMS object Object to add to this jam

on_conflict: str, default='fail'

Strategy for resolving metadata conflicts; one of ['fail', 'overwrite', or 'ignore'].

Utility functions for parsing datasets.

pyjams.util.**read_lab**(*filename*, *num_columns*, *delimiter=None*, *comment='#'*, *header=False*)
Read the rows of a labfile into memory.

An effort is made to infer datatypes, and therefore numerical values will be mapped to ints / floats accordingly.

Note: Any row with fewer than *num_columns* values will be back-filled with empty strings.

filename [str] Path to a labfile.

num_columns [int] Number of columns in lab file.

delimiter [str] lab file delimiter

comment [str] lab file comment character

header [bool] if true, the first line will be skipped

columns [list of lists] Columns of data in the labfile.

pyjams.util.**load_textlist**(*filename*)
Return a list of lines in a text file.

pyjams.util.**expand_filepaths**(*base_dir*, *rel_paths*)
Expand a list of relative paths to a give base directory.

pyjams.util.**smkdirs**(*dpath*)
Safely make a directory path if it doesn't exist.

pyjams.util.**filebase**(*filepath*)
Return the extension-less basename of a file path.

pyjams.util.**find_with_extension**(*in_dir*, *ext*, *depth=3*)
Naive depth-search into a directory for files with a given extension.

in_dir [str] Path to search.

ext [str] File extension to match.

depth [int] Depth of directories to search.

matched [list] Collection of matching file paths.

pyjams.util.**fill_observation_annotation_data**(*values*, *confidences*, *secondary_values*, *observation_annotation*)
Add a collection of data to an event annotation (in-place).

value: list of values Obervation values.

confidence: list The corresponding confidence values for each event.

secondary_value: list of values Secondary observation values.

observation_annotation: ObservationAnnotation An instantiated observation annotation to populate.

pyjams.util.**fill_event_annotation_data**(*times*, *labels*, *event_annotation*)
Add a collection of data to an event annotation (in-place).

times: list of scalars Event times in seconds.

labels: list The corresponding labels for each event.

event_annotation: EventAnnotation An instantiated event annotation to populate.

pyjams.util.**fill_range_annotation_data**(*start_times*, *end_times*, *labels*, *range_annotation*)
Add a collection of data to a range annotation (in-place).

start_times: list of scalars Start times of each range, in seconds.

end_times: list of scalars End times of each range, in seconds.

labels: list The corresponding labels for each range.

range_annotation: RangeAnnotation An instantiated range annotation to populate.

pyjams.util.**fill_timeseries_annotation_data**(*times*, *values*, *confidences*, *timeseries_annotation*)
Add a collection of data to a time-series annotation (in-place).

times: list of scalars Time points in seconds.

values: list The corresponding values for each time point.

confidences: list The corresponding confidence for each time point.

timeseries_annotation: TimeSeriesAnnotation An instantiated event annotation to populate.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

p