Auton Lab Universal Data

Anthony Wertz

Feb 27, 2021

CONTENTS

This is documentation for the *audata* package, a generic data storage schema built on top of HDF5. The code can be found on Github.

CHAPTER

ONE

OVERVIEW

The audata library is intended to standardize time series data storage and access around a specification that is easy to understand, well-supported on all major operating systems and most popular programming languages, and supports advanced features like data-streaming and compression out of the box. To this end we've developed an interface to sit on top of the HDF5 file format to promote ease of data conversion and retrieval in a consistent and well-defined format.

The *audata* specification is a generic data format schema built on top of HDF5 with the purpose of extending the supported data types above what is built in to HDF5, and to overlay various helper functions relevant to time series data. Though time series are not the only support data types, they're the primary focus for this library. *audata* is built to support efficient and pain-free streaming of data in real-time as well as indexing of very large datasets for visualization and analytic tasks.

The package is *pip*-installable:

```
pip install audata
```

CHAPTER

TWO

TUTORIAL

Coming soon!

CHAPTER

THREE

FORMAT SPECIFICATION

Here the underlying data format is explicated. The format can easily be read in without the *audata* package, although it certainly makes things easier.

3.1 HDF5 Configuration

HDF5 was chosen to support compression and streaming. For streaming support chunked datasets are used, and gzip compression is used for portability.

3.2 Data Heirarchy

At a high level, *audata* doesn't enforce any particular data heirarchy, with the exception that groups prefixed with a period are considered meta-data or ignored. All other datasets are assumed to be valid data from which time series may be extracted. Meta-data in the file is largely stored as JSON text for ease of processing.

3.3 Global Meta

At the data root there is *.meta* directory with some high-level configuration, and where string data are stored (similar to strings and cell-arrays in MATLAB's MAT files). There are two attributes here: *audata* and *data*. *audata* stores some information about the software used to generate the file and the spec version, e.g.,

```
.meta/audata
    {
        # Software version.
        "version": [20, 2, 15, 1901],

        # Specification version.
        "data_version": 1
    }
```

data stores some information about the particular file, most importantly the time reference:

```
.meta/data
    {
        "title": "...",
        "author": "...",
        "organization": "...",
        "time": {
```

(continues on next page)

(continued from previous page)

```
            "origin": "2020-41-17 15:41:22.306880 EST",
            "units": "seconds"
        }
    }
```

Anything can be stored in data, but the most important entry that _must_ be present is the time. All timestamps in the audata format are stored as offsets from the origin specified in the meta, so recovery of a specific time or date is only possible if the origin is present. The units are also important as they indicate the units for timestamps and time deltas. These can be overridden at the dataset level, but if missing the global offset and units are assumed.

3.4 String Storage

String columns are stored as variable-length strings on HDF5's heap. Unfortunately this means they do not get compressed, but this is an issue that will be resolved in the future.

Note that factor (or categorical) columns are generally not treated the same way. Read about special data types below.

3.5 Groups

Groups may store meta data to override the time offset. Otherwise there are no restrictions on groups or heirarchy. For specific applications a common schema may be desired (e.g., unifiying medical datasets and naming conventions) but audata doesn't impose any specific restrictions.

3.6 Datasets

Outside of the *.meta* group, any dataset is considered actual data. A dataset generally consists of at least two columns: a time and a value. However, neither is technically required. The usual assumption is that the first time column (usually the first column, usually named time) is present and is valid for all other columns, treated as signals with the same time index. However, *audata* is flexible and supports any number of time columns. It also supports the concept of a meta column derived from others, e.g., a time range derived from two time columns denoting the start and end of the range.

3.7 Meta

Datasets contain some meta data that document the data stored and indicate how to interpret it. Comments are optional.

```
alert_dataset/.meta
    {
        "columns": {
            "start_time": {
                "type": "time"
            },
            "end_time": {
                "type": "string"
            },
            "problem": {
                "type": "factor",
```

(continues on next page)

(continued from previous page)

```
                "levels": ["Misconfigured", "Cracked", "On Fire"],
                "ordered": true
            },
            "device_temperature": {
                "type": "real"
            },
            "times_broken": {
                "type": "integer",
                "signed": false
            },
            "other_notes": {
                "type": "string"
                "comment": "This field just contains device maintenance notes."
            }
        },
        "meta-columns": {
            "alert_period": {
                "type": "time-range",
                "start": "start_time",
                "end": "end_time"
            }
        }
    }
```

3.8 Special types

Dates and times are stored as double-precision values with units and origin specified in *.meta/data* as indicated before. When stored or retrieved using the python library, this conversion happens seamlessly. This storage mechanism is inefficient for high-density, uniformly sampled data, which is planned to be supported in the future.

Time deltas are also supported, also stored as double-precision in the indicated units. The designation is somewhat primarily for documentation but also allows automatic conversion to a timedelta object in python.

Strings were discussed previously.

Factor (or categorical) variables, even if the labels are strings, are treated differently. An integral index column is used to index into an array of factor levels, which are stored in the meta, not in the strings meta group. Automatic conversion happens when storing or reading via the python library.

The assumed difference between strings and factors is that factors are much lower arity than the size of the dataset (i.e., there are lots of repeats) versus free-text which is often unique to each entry. Typically, the number is quite small (say, less than 100) and the labels are short. This isn't always the case: a city might be a categorical, and there are many of those. But generally this assumption is reasonable, so when inferring datatypes while converting from other data sources the columns are usually treated as factor values if the arity is less than 10% of the data size, otherwise treated as strings.

3.9 Meta columns

Certain meta columns are supported. The example above is the time range, which specifies two other columns as the start and end times. Meta columns can be interpreted as hints to higher-level structure in an otherwise flat table, or (as in the case of a time range) actually be interpretted as a useable data type (e.g., plotting time ranges with shading on a graph).

Currently just time ranges are supported as meta columns. Ranges specify columns with start and end times.

CHAPTER

FOUR

API REFERENCE

4.1 audata

4.1.1 audata package

Subpackages

audata.bin package

Submodules

audata.bin.csv2audata module

Tool for automagically converting CSV files to audata files.

audata.bin.csv2audata.**main**()

Converts CSV files or nested directories containing multiple CSVs to an audata HDF5 file.

This tool takes in the path to either a single CSV file or a directory to be recursed for all CSV files. For each, it attempts to intelligently convert each CSV into an *audata* dataset and store the output to a file (with the same name as the CSV or directory, ending in '.h5').

The tool will infer a datetime column if it is named time, timestamp, or index, or if it is a string column whose first element can be parsed as a datetime object using *dateutils.parser.parse*. This parsing does rely on the first row being valid.

Other string columns will be treated either as factor/categorical columns if the arity (unique values) is less than 10% of the total number of rows, otherwise a string column.

Parameters **path** (*str*) – Path to a CSV file or directory to be recursively scanned for CSVs.

Module contents

Submodules

audata.annotation module

audata.dataset module

Classes for wrapping HDF5 datasets.

class audata.dataset.**Dataset**(*au_parent*, *name*)
Bases: *audata.element.Element*

Maps to an HDF5 dataset, maintaining the *audata* schema and facilitating translation of higher-level data types. Generally should not be instantiated directly.

Parameters

- **au_parent** (audata.element.Element) –
- **name** (*str*) –

append(*data*, *direct=False*, *time_cols=None*, *timedelta_cols=None*)
Append additional data to a dataset.

Parameters

- **data** (*Union[pandas.core.frame.DataFrame, numpy.recarray]*) –
- **direct** (*bool*) –
- **time_cols** (*Optional[AbstractSet[str]]*) –
- **timedelta_cols** (*Optional[AbstractSet[str]]*) –

property columns
Get dictionary of column specifications.

get(*idx=slice(None, - 1, None)*, *raw=False*, *datetimes=None*)
Return a dataset as a pandas DataFrame.

Parameters

- **raw** (*bool*) –
- **datetimes** (*Optional[bool]*) –

Return type pandas.core.frame.DataFrame

property ncol
Number of columns in dataset.

classmethod new(*au_parent*, *name*, *value*, *overwrite=False*, ***kwargs*)
Create a new Dataset object.

Parameters

- **au_parent** (audata.element.Element) –
- **name** (*str*) –
- **value** (*Union[h5py._hl.dataset.Dataset, numpy.ndarray, numpy.recarray, pandas.core.frame.DataFrame]*) –
- **overwrite** (*bool*) –

Return type *audata.dataset.Dataset*

property nrow
Number of rows in dataset.

property shape
Get dataset shape tuple (rows, cols).

audata.element module

Base element class.

class audata.element.**Element**(*parent=None*, *name=''*)
Bases: object

Represents an abstract *audata* element (e.g., files, groups, etc..) It should not be necessary to interact with this class directly.

Parameters

- **parent** (*Optional[Union[h5py._hl.files.File, Element]]*) –
- **name** (*str*) –

clear()
Clear members to reset class.

property filename
Path to file (*Optional[str]*, read-only)

property hdf
Get wrapped HDF object.

property meta
Element meta data (HDF5 .meta attribute) (JSON dictionary)

property meta_audata
File audata meta, '.meta/audata' attribute (JSON dictionary, read-only)

property meta_data
File data meta, '.meta/data' attribute (JSON dictionary, read-only)

property name
Element name (*Optional[str]*, read-only)

property time_reference
File time reference (*Optional[dt.datetime]*, read-only)

property valid
Is element valid? (*bool*, read-only)

audata.file module

HDF5 file wrapper class.

class audata.file.**File**(*file*, *time_reference=None*, *return_datetimes=True*)
Bases: *audata.group.Group*

Wrapper around an HDF5 file.

The wrapper adds a lot of convenience in handling audata files by automatically maintaining the correct underlying data schema while providing an intuitive interface and some convenience functions. Datasets can be accessed and updated by using a file object as a dictionary, a dictionary of dictionaries, or a dictionary where hierarchy is implied by use of the forward-slash as a "directory" delimiter, similar to how datasets are accessed using h5py. Data conversions to store higher-level data types unsupported natively by HDF5 (e.g., timestamps, ranges, or categorical variables) is handled implicitely.

Generally, files are opened or created using the *open* or *new* class methods, respectively, instead of the constructor.

Example

Creating a new file and adding a dataset:

```
>>> f = audata.File.new('test.h5', time_reference=dt.datetime(2020, 5, 4,␣
↪tzinfo=UTC))
>>> f['data'] = pd.DataFrame(data={
...     'time': f.time_reference + dt.timedelta(hours=1)*np.arange(3),
...     'a': [1,2,3],
...     'b': pd.Categorical(['a', 'b', 'c'])})
>>> f['data']
/data: Dataset [3 rows x 3 cols]
  time: time
  a: integer (signed)
  b: factor with 3 levels [a, b, c]
>>> f['data'][:]
                       time  a  b
0 2020-05-04 00:00:00+00:00  1  a
1 2020-05-04 01:00:00+00:00  2  b
2 2020-05-04 02:00:00+00:00  3  c
```

Instantiates the File object.

Generally *new* or *open* will be called, the constructor is not called directly.

Parameters

- **file** (*h5py._hl.files.File*) – The opened HDF5 file object.
- **time_reference** (*Optional[datetime.datetime]*) – The file-level time reference.
- **return_datetimes** (*bool*) – True if timestamps should be converted to *dt.datetime* objects, False if Unix timestamps (UTC) should be returned instead.

DateTimeFormat = '%Y-%m-%d %H:%M:%S.%f %Z'

close()
Close the file handle.

flush()
Flush changes to disk.

classmethod new(*filename*, *overwrite=False*, *time_reference='now'*, *title=None*, *author=None*, *organization=None*, *return_datetimes=True*, ***kwargs*)
Create a new file.

Parameters

- **filename** (*str*) – The path of the file to open.
- **overwrite** (*bool*) – If True, existing files will be truncated. Otherwise, an existing file will cause an exception.
- **time_reference** (*Union[Literal[now], datetime.datetime]*) – The time reference to use, or 'now' to use the time of file creation.
- **title** (*Optional[str]*) – A title for the dataset.
- **author** (*Optional[str]*) – The dataset author.
- **organization** (*Optional[str]*) – The dataset organization.

- **return_datetimes** (*bool*) – If True times will be converted to *dt.datetime* objects, otherwise Unix (UTC) timestamps.
- ****kwargs** – Additional keyword arguments will be passed on to *h5.File*'s constructor.

Returns The newly opened file object.

Return type *audata.file.File*

classmethod open(*filename*, *create=False*, *readonly=True*, *return_datetimes=True*, ***kwargs*)
Open an audata file.

Parameters

- **filename** (*str*) – The path to the file to open.
- **create** (*bool*) – If True, missing files will be created. Otherwise, missing files cause an exception.
- **readonly** (*bool*) – Whether to open in read-only or mutable.
- **return_datetimes** (*bool*) – If True times will be converted to *dt.datetime* objects, otherwise Unix (UTC) timestamps.
- ****kwargs** – Additional keyword arguments will be passed on to *h5.File*'s constructor if a file is to be created.

Returns The opened file object.

Return type *audata.file.File*

property time_reference
The timezone-aware time reference.

Can be set with either a *dt.datetime* object or a *str* that can be parsed as a datetime. If a naive datetime is provided, the local timezone will be inferred.

audata.group module

Wrapper for Group types.

class audata.group.**Group**(*au_parent*, *name=''*)
Bases: *audata.element.Element*

Group element. Acts largely like a container for datasets. Generally should not be instantiated directly.

Parameters

- **au_parent** (audata.element.Element) –
- **name** (*str*) –

list()
List all child attributes, groups, and datasets.

Return type Dict[str, List[str]]

new_dataset(*name*, *value*, ***kwargs*)
Create a new dataset.

Parameters

- **name** (*str*) –
- **value** (*Union[h5py._hl.dataset.Dataset, np.ndarray, np.recarray, pd.DataFrame]*) –

recurse()
Recursively find all datasets. Groups and datasets prefixed with a period are ignored.

Returns Element, name: str).

Return type Iterable (generator) of tuples of (object

Module contents

Define version and import high-level objects.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX