DSCleaner

Release 0.1.7

May 11, 2021

Package contents:

CHAPTER 1

What is DS Cleaner?

DS Cleaner (Dataset Cleaner) is a library that allows for easy cleanup of energy disagregation datasets and allows converting in wave and wave64. Is based in librosa and PySoundFile library.

1.1 DS Cleaner Package

1.1.1 DSCleaner's class diagram

These are the main components of the package

1.1.2 dscleaner.IFileInfo class

class `dscleaner.ifileinfo.`**`IFileInfo`**`(`*file*`)`

Bases: `abc.ABC`

Interface which must be implemented if you want to support your own filetype; Used as an argument to FileWriter, FileUtil and FileMerger.

`addSamples``(`*samples*`)`

Adds samples given by `samples`.

Parameters **`samples`** – An array containing samples. It must be shaped like (n,c) where c is the number of channels.

`close``()`

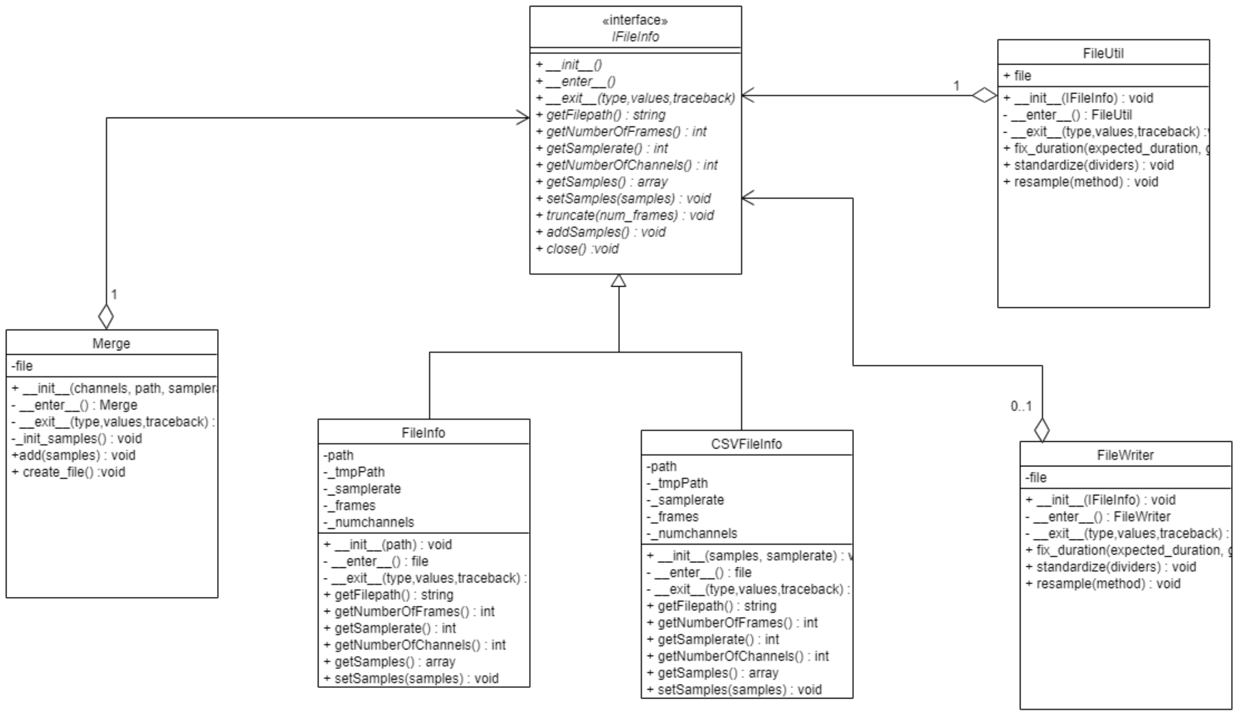
Defines the behavior the class should have when leaves the context manager.

If a file descriptor is being used you should always define the close method.

`getFilepath``()`

Returns The filepath of the current file.

`getNumberOfChannels``()`


«interface»
IFileInfo
+ __init__()
+ __enter__()
+ __exit__(type,values,traceback)
+ getFilepath() : string
+ getNumberOfFrames() : int
+ getSamplerate() : int
+ getNumberOfChannels() : int
+ getSamples() : array
+ setSamples(samples) : void
+ truncate(num_frames) : void
+ addSamples() : void
+ close() :void
FileUtil
+ file
+ __init__(IFileInfo) : void
- __enter__() : FileUtil
- __exit__(type,values,traceback) :
+ fix_duration(expected_duration,
+ standardize(dividers) : void
+ resample(method) : void
1
Merge
-file
+ __init__(channels, path, sampler
- __enter__() : Merge
- __exit__(type,values,traceback) :
-_init_samples() : void
+add(samples) : void
+ create_file() :void
1
FileInfo
-path
-_tmpPath
-_samplerate
-_frames
-_numchannels
+ __init__(path) : void
- __enter__() : file
- __exit__(type,values,traceback) :
+ getFilepath() : string
+ getNumberOfFrames() : int
+ getSamplerate() : int
+ getNumberOfChannels() : int
+ getSamples() : array
+ setSamples(samples) : void
CSVFileInfo
-path
-_tmpPath
-_samplerate
-_frames
-_numchannels
+ __init__(samples, samplerate) :
- __enter__() : file
- __exit__(type,values,traceback) :
+ getFilepath() : string
+ getNumberOfFrames() : int
+ getSamplerate() : int
+ getNumberOfChannels() : int
+ getSamples() : array
+ setSamples(samples) : void
0..1
FileWriter
-file
+ __init__(IFileInfo) : void
- __enter__() : FileWriter
- __exit__(type,values,traceback) :
+ fix_duration(expected_duration,
+ standardize(dividers) : void
+ resample(method) : void

Returns The number of channels the file has.

getNumberOfFrames()

Returns The number of frames the file has.

getSamplerate()

Returns The samplerate of the file.

getSamples()

Returns All the samples the file has.

setSamples(*samples*)

Writes to the file the samples given in `samples`.

It will truncate the old samples. If you want to add, use the `addSamples` method.

Parameters **samples** – An array containing the new samples. It must be shaped like (n,c) where c is the number of channels

truncate(*num_frames*)

Truncates the file to have only the first `num_frames` samples.

Parameters **num_frames** – The number of frames the file will have.

1.1.3 dscleaner.CSVFileInfo class

class dscleaner.csvfileinfo.**CsvFileInfo**(*samples*, *samplerate*)

Bases: *dscleaner.ifileinfo.IFileInfo*

CsvFileInfo is used when there is no actual file, but an array.

The array must be shaped in (n,c) where c is the number of channels.

addSamples(*samples*)
    See the base class `ifileinfo`.

close()
    Does nothing since the sample container is an array.

getFilepath()
    Don't rely on this method because it's not implemented.

getNumberOfChannels()
    See the base class `ifileinfo`.

getNumberOfFrames()
    See the base class `ifileinfo`.

getSamplerate()
    See the base class `ifileinfo`.

getSamples()
    See the base class `ifileinfo`.

setSamples(*samples*, *framerate=None*)
    See the base class `ifileinfo`.

truncate(*num_frames*)
    See the base class `ifileinfo`.

1.1.4 dscleaner.FileInfo class

class dscleaner.fileinfo.**FileInfo**(*path*)
    Bases: *dscleaner.ifileinfo.IFileInfo*

Defines the class to manipulate soundfiles.

Receives a path to a file.

Copies the file to a temporary location.

Gets edited through the FileUtil.

FileWriter converts and writes to another location.

Note:

- close method MUST always be called or else the temporary file stays in disk.
- `with` statements should be used in order to close the files automatically.

addSamples(*samples*)
    Appends the samples to the file.

    Similar to ´setSamples()´ but appends instead of truncating. See the base class `ifileinfo`.

close()
    Must always be called or the file won't be accessible by other processes AND the temp file will stay in disk. See the base class `ifileinfo`.

getDuration()

    Returns The duration of the file in seconds.

getFilepath()
See the base class `ifileinfo`.

getNumberOfChannels()

Returns Number of channels the file has.

getNumberOfFrames()
See the base class `ifileinfo`.

getSamplerate()
See the base class `ifileinfo`.

getSamples()
Reads all of the samples in the file

Returns numpy array containing the samples.

get_rounded_duration()

Returns The rounded duration in seconds.

setSamples(*samples*, *samplerate=None*)
Writes the ´samples´ as the new samples in the file.

Parameters

- **samples** – numpy array shaped like (n,c), where c is the number of channels.
- **samplerate** – the new sample rate the file will have, if none it will use the initial samplerate.

truncate(*num_frames*)
Truncates the file to only have `num_frames`.

1.1.5 dscleaner.FileUtil class

class dscleaner.fileutil.**FileUtil**(*f*)
Bases: `object`

FileUtil class is where the dataset manipulation occur.

The class should be instantiated with a `with` statement.

Parameters **f** – a IFileInfo specialization must be supplied!

fix_duration(*expected_duration*, *grid_rate=50*)
Fixes the file to the expected duration.

Parameters

- **expected_duration** – Duration the file should have in minutes.
- **grid_rate** – frequency of the grid in hertz, this is used to discover the wave signal in order to upsample.

resample(*new_framerate*, *method='kaiser_fast'*)
Resamples the data to the new framerate using librosa resample.

Parameters

- **data** – numpy.array shaped like (num_frames,num_channels) is expected to receive the soundfile.getSamples() not the transposed array.
- **original_framerate** – the original framerate the data array uses.

- **new_framerate** – the new framerate that data will be resampled to.
- **method** – Methods that librosa accepts are also accepted here, uses *kaiser_fast* by default.

standardize(**dividers*)

This method transforms the values to fit between -1 and 1, in order to be used in soundfiles.

If the source file isn't a soundfile the target file will not be well formated, hence you should run this method to make the file well formated.

Parameters ***dividers** – The number which each channel will be divided by in order to standardize that channel.

Note: In order to maintain consistency throughout the dataset it is advised that the divider chosen for each channel to be a bit higher than the max value. It is also advised to keep record of the divider for each channel for future unstardartization.

Example: Max amplitude is 75 divider chosen: 90.

Returns

A tuple with the dividers used to standardize.

Example: (40,30,30) in a three channel file.

You should keep these values for future reference.

1.1.6 dscleaner.FileWriter class

class `dscleaner.filewriter.`**FileWriter**(*file*, *mode='w'*)

Bases: `object`

Writes to a file.

The class should be instantiated with a `with` statement.

Parameters

- **file** – Accepts either a FileUtil, or IFileInfo Specialization.
- **mode** – Allows for *w* for writing or *a* for appending.

close()

create_file(*new_filepath*, *samplerate=None*)

Creates a new file with the extension given in `new_filepath`.

If the source file isn't a soundfile the target file will not be well formated.

In order to normalize, you should run `FileUtil.standardize` method before.

Parameters

- **– The diretory and name the new file will have,** (*new_filepath*) – it will convert based on file extension.
- **samplerate** (*Optional*) – if not supplied it will use the own *file* samplerate.

create_file_EMDDF(*new_filepath*, *json_file*, *samplerate=None*)

Creates a soundfile with the EMD-DF format, recurs to the `pyemddf` package

Note: Only works on wave and wave64 files.

Parameters

- **`samplerate`** (*`optional`*) – Samplerate of the file.
- **`json_file`** – a JSON file with the metadata fields, you can get a template
- **`it by executing pyemddf.create_template_file()`** (*`for`*) –

1.1.7 dscleaner.Merger class

class `dscleaner.merger.`**`Merger`**(*channels*, *path*, *samplerate*, *cutoff=None*, *mode='a'*)

Bases: `object`

Merger allows for creation of an empty soundfile to store multiple datasets easily.

Note: W64 filetype is recommended, given it can store up to 18 exabytes of data.

Parameters

- **`channels`** – Number of channels the files parsed should have.
- **`path`** – the path where the new merger file should be created.
- **`samplerate`** – samplerate to write on the file.
- **`cutoff`** – how often should the file be written **NOT IMPLEMENTED** (eg. for each 1024MB of data reached a new file is created)
- **`mode`** – either 'a' or 'w' if the file should be appended or truncated, respectively. Default behavior: append

`add`(**files*)

Adds new samples to the buffer array.

When `create_file` method is executed the buffer gets emptied.

Parameters **`*files`** – An array, containing several pathes to files or IFileInfos specializations, although the latter is preferred.

`create_file`(*samplerate=None*)

Creates a new file with the filename, converts based on extension given in `new_filename`

When executed the sample buffer will be emptied, so `create_file` should be executed frequently.

Parameters **`samplerate`** – Samplerate of the file.

1.1.8 dscleaner.Splitter class

class `dscleaner.splitter.`**`Splitter`**(*channels*, *path*, *samplerate*, *max_length*)

Bases: `object`

Splitter allows splitting an existing file.

Parameters

- **channels** – Number of channels the files parsed should have.
- **path** – the path where the new merger file should be created.
- **samplerate** – samplerate to write on the file.
- **max_length** – Maximum file length in minutes.

add(**files*)

Adds new samples to the buffer array.

When `create_file` method is executed the buffer gets emptied.

Parameters ***files** – An array, containing several pathes to files or IFileInfos specializations, although the latter is preferred.

create_file(*samplerate=None*)

Creates a new file with the filename, converts based on extension given in `new_filename`

When executed the sample buffer will be emptied, so `create_file` should be executed frequently.

Parameters **samplerate** – Samplerate of the file.

1.1.9 dscleaner.Utils module

`dscleaner.utils.`**is_number**(*input*)

Receives an input and checks if it is actually a number

`dscleaner.utils.`**path_splitter**(*path*)

Cleans extra / characters, splits the path in 4 parts: See example

Parameters **path** – Receives a path

Returns

dictionary with the following keys: {full_path, path, file, file_name, extension}

Return type tuple

Example

```
>>> path.splitter('C:/Data/example.wav/')
{
    'full_path':'C:/Data/example.wav',
    'path':'C:/Data/',
    'file':'example',
    'file_name':'example.wav',
    'extension':'wav'
}
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index