

---

# **mirdata**

***Release 0.2.1***

**Jan 14, 2021**



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Documentation</b>	<b>5</b>
2.1	Table of supported datasets . . . . .	5
2.2	Examples . . . . .	8
2.3	API documentation . . . . .	12
2.4	FAQ . . . . .	71
<b>3</b>	<b>Contribute</b>	<b>73</b>
	<b>Python Module Index</b>	<b>75</b>
	<b>Index</b>	<b>77</b>



This library provides tools for working with common MIR datasets, including tools for:

- downloading datasets to a common location and format
- validating that the files for a dataset are all present
- loading annotation files to a common format, consistent with the format required by *mir\_eval*
- parsing track level metadata for detailed evaluations.

This library was presented in our [ISMIR 2019 paper](#)



# CHAPTER 1

---

## Installation

---

```
pip install mirdata
```





## 2.1 Table of supported datasets

This table is provided as a guide for users to select appropriate datasets. The list of annotations omits some metadata for brevity, and we document the dataset’s primary annotations only. The number of tracks indicates the number of unique “tracks” in a dataset, but it may not reflect the actual size or diversity of a dataset, as tracks can vary greatly in length (from a few seconds to a few minutes), and may be homogeneous. For specific information about the contents of each dataset, click the link provided in the “Module” column.

“Downloadable” possible values:

- : Freely downloadable
- : Available upon request
- : Youtube Links only
- : Not available

Module	Name	Downloadable?	Annotation Types	Tracks
beatles	The Beatles Dataset	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li><i>Beats</i></li> <li><i>Sections</i></li> <li><i>Key</i></li> </ul>	180
beatport_key	Beatport EDM key	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li>global <i>Key</i></li> </ul>	1486
dali	DALI	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li><i>Lyrics</i></li> <li>Vocal <i>Notes</i></li> </ul>	5358
groove_midi	Groove MIDI Dataset	<ul style="list-style-type: none"> <li>audio:</li> <li>midi:</li> </ul>	<ul style="list-style-type: none"> <li><i>Beats</i></li> <li><i>Tempo</i></li> <li><i>Drums</i></li> </ul>	1150
gtzan_genre	Gtzan-Genre	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li><i>Genre</i></li> </ul>	1000
giantsteps_tempo	Giantsteps EDM tempo Dataset	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li><i>Genre</i></li> <li><i>Tempo</i></li> </ul>	664
giantsteps_key	Giantsteps EDM key	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li>global <i>Key</i></li> </ul>	500
guitarset	GuitarSet	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li><i>Beats</i></li> <li><i>Chords</i></li> <li><i>Key</i></li> <li><i>Notes</i></li> <li><i>F0</i></li> <li><i>Tempo</i></li> </ul>	360
ikala	iKala	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li>Vocal <i>F0</i></li> <li><i>Lyrics</i></li> </ul>	252
maestro	MAESTRO	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li>Piano <i>Notes</i></li> </ul>	1282
medley_solos_db	Medley-solos-DB	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li><i>Instruments</i></li> </ul>	21571
medleydb_melody	MedleyDB Melody Subset	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li><i>Melody F0</i></li> </ul>	108
medleydb_pitch	MedleyDB Pitch Tracking Subset	<ul style="list-style-type: none"> <li>audio:</li> <li>annotations:</li> </ul>	<ul style="list-style-type: none"> <li><i>F0</i></li> <li><i>Instruments</i></li> </ul>	103
orchset	Orchset	<ul style="list-style-type: none"> <li>audio:</li> </ul>	<ul style="list-style-type: none"> <li><i>Melody F0</i></li> </ul>	64

### 2.1.1 Annotation Type Descriptions

The table above provides annotation types as a guide for choosing appropriate datasets, but it is difficult to generically categorize annotation types, as they depend on varying definitions and their meaning can change depending on the type of music they correspond to. Here we provide a rough guide to the types in this table, but we **strongly recommend** reading the dataset specific documentation to ensure the data is as you expect.

#### Beats

Musical beats, typically encoded as sequence of timestamps and corresponding beat positions. This implicitly includes *downbeat* information (the beginning of a musical measure).

#### Chords

Musical chords, e.g. as might be played on a guitar. Typically encoded as a sequence of labeled events, where each event has a start time, end time, and a label. The label taxonomy varies per dataset, but typically encode a chord's root and its quality, e.g. A:m7 for "A minor 7".

#### Drums

Transcription of the drums, typically encoded as a sequence of labeled events, where the labels indicate which drum instrument (e.g. cymbal, snare drum) is played. These events often overlap with one another, as multiple drums can be played at the same time.

#### F0

Musical pitch contours, typically encoded as time series indicating the musical pitch over time. The time series typically have evenly spaced timestamps, each with a corresponding pitch value which may be encoded in a number of formats/granularities, including midi note numbers and Hertz.

#### Genre

A typically global "tag", indicating the genre of a recording. Note that the concept of genre is highly subjective and we refer those new to this task to this [article](#).

#### Instruments

Labels indicating which instrument is present in a musical recording. This may refer to recordings of solo instruments, or to recordings with multiple instruments. The labels may be global to a recording, or they may vary over time, indicating the presence/absence of a particular instrument as a time series.

#### Key

Musical key. This can be defined globally for an audio file or as a sequence of events.

## Lyrics

Lyrics corresponding to the singing voice of the audio. These may be raw text with no time information, or they may be time-aligned events. They may have varying levels of granularity (paragraph, line, word, phoneme, character) depending on the dataset.

## Melody

The musical melody of a song. Melody has no universal definition and is typically defined per dataset. It is typically encoded as *F0* or as *Notes*. Other types of annotations such as Vocal F0 or Vocal Notes can often be considered as melody annotations as well.

## Notes

Musical note events, typically encoded as sequences of start time, end time, label. The label typically indicates a musical pitch, which may be in a number of formats/granularities, including midi note numbers, Hertz, or pitch class.

## Sections

Musical sections, which may be “flat” or “hierarchical”, typically encoded by a sequence of timestamps indicating musical section boundary times. Section annotations sometimes also include labels for sections, which may indicate repetitions and/or the section type (e.g. Chorus, Verse).

## Technique

The playing technique used by a particular instrument, for example “Pizzicato”. This label may be global for a given recording or encoded as a sequence of labeled events.

## Tempo

The tempo of a song, typical in units of beats-per-minute (bpm). This is often indicated globally per track, but in practice tracks may have tempos that change, and some datasets encode tempo as time-varying quantity. Additionally, there may be multiple reasonable tempos at any given time (for example, often 2x or 0.5x a tempo value will also be “correct”). For this reason, some datasets provide two or more different tempo values.

## Vocal Activity

A time series or sequence of events indicating when singing voice is present in a recording. This type of annotation is implicitly available when Vocal *F0* or Vocal *Notes* annotations are available.

# 2.2 Examples

## 2.2.1 Basic Example

First of all, you can install mirdata using *pip*.

```
$ pip install mirdata
```

With *mirdata*, the first thing you would do is to download the dataset. Depending on the availability of the audio files, *mirdata* may only show you an instruction about how to download the dataset. Fortunately, we can download Orchset dataset directly.

```
1 import mirdata.orchset
2 # Download the Orchset Dataset
3 mirdata.orchset.download()
4 # Orchset_dataset_0.zip?download=1: 1.00B [03:05, 185s/B]
```

Once downloading is done, you can find the the dataset folder.

```
1 $ ls ~/mir_datasets/Orchset/
2 GT
3 Orchset - Predominant Melodic Instruments.csv
4 README.txt
5 audio
6 midi
```

The ID's and annotation data can be loaded as below.

```
1 # Load the dataset
2 orchset_data = mirdata.orchset.load()
3 orchset_ids = mirdata.orchset.track_ids()
4
5 # todo: add __str__() method and print(orchset_data)
```

If we wanted to use Orchset to evaluate the performance of a melody extraction algorithm (in our case, *very\_bad\_melody\_extractor*), and then split the scores based on the metadata, we could do the following:

```
1 import mir_eval
2 import mirdata.orchset
3 import numpy as np
4 import sox
5
6 def very_bad_melody_extractor(audio_path):
7     duration = sox.file_info.duration(audio_path)
8     time_stamps = np.arange(0, duration, 0.01)
9     melody_f0 = np.random.uniform(low=80.0, high=800.0, size=time_stamps.shape)
10    return time_stamps, melody_f0
11
12 # Evaluate on the full dataset
13 orchset_scores = {}
14 orchset_data = mirdata.orchset.load()
15 for track_id, track_data in orchset_data.items():
16     est_times, est_freqs = very_bad_melody_extractor(track_data.audio_path_mono)
17
18     ref_melody_data = track_data.melody
19     ref_times = ref_melody_data.times
20     ref_freqs = ref_melody_data.frequencies
21
22     score = mir_eval.melody.evaluate(ref_times, ref_freqs, est_times, est_freqs)
23     orchset_scores[track_id] = score
24
25 # Split the results by composer and by instrumentation
26 composer_scores = {}
27 strings_no_strings_scores = {True: {}, False: {}}
28 for track_id, track_data in orchset_data.items():
29     if track_data.composer not in composer_scores.keys():
```

(continues on next page)

(continued from previous page)

```

30     composer_scores[track_data.composer] = {}
31
32     composer_scores[track_data.composer][track_id] = orchset_scores[track_id]
33     strings_no_strings_scores[track_data.contains_strings][track_id] = \
34         orchset_scores[track_id]

```

This is the result of the example above.

```

1  # strings_no_strings_scores
2
3  {True: {'Beethoven-S3-I-ex1': OrderedDict([('Voicing Recall', 1.0),
4         ('Voicing False Alarm', 1.0),
5         ('Raw Pitch Accuracy', 0.029798422436459245),
6         ('Raw Chroma Accuracy', 0.08063102541630149),
7         ('Overall Accuracy', 0.0272654370489174)]),
8        'Beethoven-S3-I-ex2': OrderedDict([('Voicing Recall', 1.0),
9         ('Voicing False Alarm', 1.0),
10         ('Raw Pitch Accuracy', 0.009221311475409836),
11         ('Raw Chroma Accuracy', 0.07377049180327869),
12         ('Overall Accuracy', 0.008754863813229572)]),
13
14  ...
15
16  'Wagner-Tannhauser-Act2-ex2': OrderedDict([('Voicing Recall', 1.0),
17         ('Voicing False Alarm', 1.0),
18         ('Raw Pitch Accuracy', 0.03685636856368564),
19         ('Raw Chroma Accuracy', 0.08997289972899729),
20         ('Overall Accuracy', 0.036657681940700806)]})}

```

*very\_bad\_melody\_extractor* performs very badly!

## 2.2.2 Using mirdata with local vs. remote data

When using mirdata on the same machine as where your datasets live, we do the loading for you.

For example, to load the melody annotations from Orchset into memory, we can simply call:

```

1  import mirdata.orchset
2
3  # Load a single track
4  track = mirdata.orchset.Track('Beethoven-S3-I-ex1')
5  melody_annotation = track.melody
6
7  print(melody_annotation)
8  # F0Data(times=array([0.000e+00, 1.000e-02, 2.000e-02, ..., 1.244e+01, 1.245e+01,
9  #    1.246e+01]), frequencies=array([ 0. ,  0. ,  0. , ..., 391.995, 391.995,
10  ↪ 391.995]), confidence=array([0., 0., 0., ..., 1., 1., 1.]))

```

However, if your data lives somewhere else, accessing the annotation will return *None*. Instead, you can use the module's loading functions directly:

```

1  import mirdata.orchset
2
3  # Load a single track, specifying the remote location
4  track = mirdata.orchset.Track('Beethoven-S3-I-ex1', data_home='gs://my_custom/remote_
5  ↪path')

```

(continues on next page)

(continued from previous page)

```

5 melody_path = track.melody_path
6
7 print(melody_path)
8 # gs://my_custom/remote_path/GT/Beethoven-S3-I-ex1.mel
9 print(os.path.exists(melody_path))
10 # False
11
12 # write code here to locally download your path e.g. to a temporary file.
13 def my_downloader(remote_path):
14     # the contents of this function will depend on where your data lives, and how
15     # permanently you want the files to remain on the machine. We point you to libraries
16     # handling common use cases below.
17     # for data you would download via scp, you could use the [scp](https://pypi.org/
18     # project/scp/) library
19     # for data on google drive, use [pydrive](https://pythonhosted.org/PyDrive/)
20     # for data on google cloud storage use [google-cloud-storage](https://pypi.org/
21     # project/google-cloud-storage/)
22     return local_path_to_downloaded_data
23
24 temp_path = my_downloader(melody_path)
25
26 # call orchset's melody annotation loader
27 melody_annotation = orchset.load_melody(temp_path)
28
29 print(melody_annotation)
30 # F0Data(times=array([0.000e+00, 1.000e-02, 2.000e-02, ..., 1.244e+01, 1.245e+01,
31 # 1.246e+01]), frequencies=array([ 0. ,  0. ,  0. , ..., 391.995, 391.995,
32 # 391.995]), confidence=array([0., 0., 0., ..., 1., 1., 1.]))

```

## 2.2.3 Using mirdata with tf.data.Dataset

The following is a simple example of a generator that can be used to create a tensorflow Dataset

```

1 import mirdata.orchset
2 import numpy as np
3 import tensorflow as tf
4
5 def orchset_generator():
6     # using the default data_home
7     track_ids = mirdata.orchset.track_ids()
8     for track_id in track_ids:
9         track = mirdata.orchset.Track(track_id)
10        audio_signal, sample_rate = track.audio_mono
11        yield {
12            "audio": audio_signal.astype(np.float32),
13            "sample_rate": sample_rate,
14            "annotation": {
15                "times": track.melody.times.astype(np.float32),
16                "freqs": track.melody.frequencies.astype(np.float32),
17            },
18            "metadata": {"track_id": track.track_id}
19        }
20
21 dataset = tf.data.Dataset.from_generator(
22     orchset_generator,

```

(continues on next page)

(continued from previous page)

```
23 {  
24     "audio": tf.float32,  
25     "sample_rate": tf.float32,  
26     "annotation": {"times": tf.float32, "freqs": tf.float32},  
27     "metadata": {'track_id': tf.string}  
28 }  
29 )
```

## 2.3 API documentation

### 2.3.1 Dataset Loaders

#### mirdata.beatles

Beatles Dataset Loader

The Beatles Dataset includes beat and metric position, chord, key, and segmentation annotations for 179 Beatles songs. Details can be found in [http://matthiasmauch.net/\\_pdf/mauch\\_omp\\_2009.pdf](http://matthiasmauch.net/_pdf/mauch_omp_2009.pdf) and <http://isophonics.net/content/reference-annotations-beatles>.

**class** mirdata.beatles.**Track** (*track\_id*, *data\_home=None*)

Beatles track class

#### Parameters

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

#### **audio\_path**

track audio path

**Type** str

#### **beats\_path**

beat annotation path

**Type** str

#### **chords\_path**

chord annotation path

**Type** str

#### **keys\_path**

key annotation path

**Type** str

#### **sections\_path**

sections annotation path

**Type** str

#### **title**

title of the track

**Type** str



**track\_id**  
track id  
**Type** str

**audio**  
audio signal, sample rate  
**Type** (np.ndarray, float)

**beats**  
human-labeled beat annotation  
**Type** BeatData

**chords**  
chord annotation  
**Type** ChordData

**key**  
key annotation  
**Type** KeyData

**sections**  
section annotation  
**Type** SectionData

**to\_jams()**  
Jams: the track's data in jams format

`mirdata.beatles.cite()`  
Print the reference

`mirdata.beatles.download(data_home=None, force_overwrite=False, cleanup=True)`  
Download the Beatles Dataset (annotations). The audio files are not provided due to copyright issues.

**Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.beatles.load(data_home=None)`  
Load Beatles dataset

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.beatles.load_audio(audio_path)`  
Load a Beatles audio file.

**Parameters** **audio\_path** (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.beatles.load_beats (beats_path)`

Load Beatles format beat data from a file

**Parameters** `beats_path` (*str*) – path to beat annotation file

**Returns** loaded beat data

**Return type** (`utils.BeatData`)

`mirdata.beatles.load_chords (chords_path)`

Load Beatles format chord data from a file

**Parameters** `chords_path` (*str*) – path to chord annotation file

**Returns** loaded chord data

**Return type** (`utils.ChordData`)

`mirdata.beatles.load_key (keys_path)`

Load Beatles format key data from a file

**Parameters** `keys_path` (*str*) – path to key annotation file

**Returns** loaded key data

**Return type** (`utils.KeyData`)

`mirdata.beatles.load_sections (sections_path)`

Load Beatles format section data from a file

**Parameters** `sections_path` (*str*) – path to section annotation file

**Returns** loaded section data

**Return type** (`utils.SectionData`)

`mirdata.beatles.track_ids ()`

Get the list of track IDs for this dataset

**Returns** A list of track ids

**Return type** (`list`)

`mirdata.beatles.validate (data_home=None, silence=False)`

Validate if a local version of this dataset is consistent

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths where the expected file exists locally but has a different checksum than the reference

**Return type** `missing_files` (`list`)

## mirdata.beatport\_key

**beatport\_key Dataset Loader** The Beatport EDM Key Dataset includes 1486 two-minute sound excerpts from various EDM subgenres, annotated with single-key labels, comments and confidence levels generously provided by Eduard Mas Marín, and thoroughly revised and expanded by Ángel Faraldo.

The original audio samples belong to online audio snippets from Beatport, an online music store for DJ's and Electronic Dance Music Producers (<<http://www.beatport.com>>). If this dataset were used in further research, we would appreciate

the citation of the current DOI (10.5281/zenodo.1101082) and the following doctoral dissertation, where a detailed description of the properties of this dataset can be found:

Ángel Faraldo (2017). Tonality Estimation in Electronic Dance Music: A Computational and Musically Informed Examination. PhD Thesis. Universitat Pompeu Fabra, Barcelona.

This dataset is mainly intended to assess the performance of computational key estimation algorithms in electronic dance music subgenres.

Data License: Creative Commons Attribution Share Alike 4.0 International

**class** `mirdata.beatport_key.Track` (*track\_id*, *data\_home=None*)  
 beatport\_key track class :param track\_id: track id of the track :type track\_id: str :param data\_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, `~/mir_datasets`

**audio\_path**  
 track audio path  
**Type** str

**keys\_path**  
 key annotation path  
**Type** str

**metadata\_path**  
 sections annotation path  
**Type** str

**title**  
 title of the track  
**Type** str

**track\_id**  
 track id  
**Type** str

**artists**  
 artist annotation  
**Type** Dict

**audio**  
 audio signal, sample rate  
**Type** (np.ndarray, float)

**genres**  
 genre annotation  
**Type** Dict

**key**  
 key annotation  
**Type** String

**tempo**  
 tempo beatports crowdsourced annotation  
**Type** int

**to\_jams()**  
 Jams: the track's data in jams format

`mirdata.beatport_key.cite()`  
 Print the reference

`mirdata.beatport_key.download(data_home=None, force_overwrite=False, cleanup=True, partial_download=None)`  
 Download the beatport\_key Dataset (annotations). The audio files are not provided due to copyright issues.

This dataset annotations have characters that doesn't correspond with json format. In particular, "bpm": nan doesn't correspond to json format. The function `find_replace` is used to fix this problem. input file :param data\_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, `~/mir_datasets`

#### Parameters

- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.
- **partial\_download** (*list of str*) – arguments can be 'audio' 'metadata' or/and 'keys'

`mirdata.beatport_key.find_replace(directory, find, replace, pattern)`

Replace in some directory all the songs with the format pattern find by replace :param directory (str) path to directory: :param find (str) string from replace: :param replace (str) string to replace: :param pattern (str) regex that must match the directories searched:

`mirdata.beatport_key.load(data_home=None)`

Load beatport\_key dataset :param data\_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.beatport_key.load_artist(metadata_path)`

Load beatport\_key tempo data from a file :param metadata\_path: path to metadata annotation file :type metadata\_path: str

**Returns** list of artists involved in the track.

**Return type** (list of strings)

`mirdata.beatport_key.load_audio(audio_path)`

Load a beatport\_key audio file. :param audio\_path: path to audio file :type audio\_path: str

**Returns** the mono audio signal sr (float): The sample rate of the audio file

**Return type** y (np.ndarray)

`mirdata.beatport_key.load_genre(metadata_path)`

Load beatport\_key genre data from a file :param metadata\_path: path to metadata annotation file :type metadata\_path: str

**Returns** with the list of strings with genres ['genres'] and list of strings with sub-genres ['sub\_genres']

**Return type** (dict)

`mirdata.beatport_key.load_key(keys_path)`

Load beatport\_key format key data from a file :param keys\_path: path to key annotation file :type keys\_path: str

**Returns** loaded key data

**Return type** (str)

`mirdata.beatport_key.load_tempo(metadata_path)`

Load beatport\_key tempo data from a file :param metadata\_path: path to metadata annotation file :type metadata\_path: str

**Returns** loaded tempo data

**Return type** (str)

`mirdata.beatport_key.track_ids()`

Get the list of track IDs for this dataset :returns: A list of track ids :rtype: (list)

`mirdata.beatport_key.validate(data_home=None, silence=False)`

Validate if a local version of this dataset is consistent :param data\_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths where the expected file exists locally but has a different checksum than the reference

**Return type** missing\_files (list)

## mirdata.dali

DALI Dataset Loader

DALI contains 5358 audio files with their time-aligned vocal melody. It also contains time-aligned lyrics at four levels of granularity: notes, words, lines, and paragraphs.

For each song, DALI also provides additional metadata: genre, language, musician, album covers, or links to video clips.

For more details, please visit: <https://github.com/gabolsgabs/DALI>

**class** `mirdata.dali.Track(track_id, data_home=None)`

DALI melody Track class

**Parameters**

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**album**

the track's album

**Type** str

**annotation\_path**  
path to the track's annotation file  
**Type** str

**artist**  
the track's artist  
**Type** str

**audio\_path**  
path to the track's audio file  
**Type** str

**audio\_url**  
youtube ID  
**Type** str

**dataset\_version**  
dataset annotation version  
**Type** int

**ground\_truth**  
True if the annotation is verified  
**Type** bool

**language**  
sung language  
**Type** str

**release\_date**  
year the track was released  
**Type** str

**scores\_manual**  
TODO  
**Type** int

**scores\_ncc**  
TODO  
**Type** float

**title**  
the track's title  
**Type** str

**track\_id**  
the unique track id  
**Type** str

**url\_working**  
True if the youtube url was valid  
**Type** bool

**annotation\_object**  
DALI Annotations object

**Type** DALI.Annotations

**audio**  
audio signal, sample rate  
**Type** (np.ndarray, float)

**lines**  
line-aligned lyrics  
**Type** LyricData

**notes**  
note-aligned lyrics  
**Type** NoteData

**paragraphs**  
paragraph-aligned lyrics  
**Type** LyricData

**to\_jams()**  
Jams: the track's data in jams format

**words**  
word-aligned lyric  
**Type** LyricData

`mirdata.dali.cite()`  
Print the reference

`mirdata.dali.download(data_home=None, force_overwrite=False)`  
DALI is not available for downloading directly. This function prints a helper message to download DALI through zenodo.org.

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

`mirdata.dali.load(data_home=None)`  
Load DALI dataset

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns** `{track_id: track data}`

**Return type** (dict)

`mirdata.dali.load_annotations_class(annotations_path)`  
Load full annotations into the DALI class object

**Parameters** `annotations_path` (*str*) – path to a DALI annotation file

**Returns** DALI annotations object

`mirdata.dali.load_annotations_granularity(annotations_path, granularity)`  
Load annotations at the specified level of granularity

**Parameters**

- `annotations_path` (*str*) – path to a DALI annotation file
- `granularity` (*str*) – one of 'notes', 'words', 'lines', 'paragraphs'

**Returns** NoteData for granularity='notes' or LyricData otherwise

`mirdata.dali.load_audio(audio_path)`

Load a DALI audio file.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.dali.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.dali.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list): List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (list)

## **mirdata.giantsteps\_tempo**

`giantsteps_tempo` Dataset Loader

name: GiantSteps (tempo+genre)

**contact:** Richard Vogl <[richard.vogl@tuwien.ac.at](mailto:richard.vogl@tuwien.ac.at)> Peter Knees <[peter.knees@tuwien.ac.at](mailto:peter.knees@tuwien.ac.at)>

**description:** collection of annotations for 664 2min(1) audio previews from [www.beatport.com](http://www.beatport.com)

**references:** [1] Peter Knees, Ángel Faraldo, Perfecto Herrera, Richard Vogl, Sebastian Böck, Florian Hörschläger, Mickael Le Goff: “Two data sets for tempo estimation and key detection in electronic dance music annotated from user corrections”, Proc. of the 16th Conference of the International Society for Music Information Retrieval (ISMIR’15), Oct. 2015, Malaga, Spain.

[2] Hendrik Schreiber, Meinard Müller: “A Crowdsourced Experiment for Tempo Estimation of Electronic Dance Music”, Proc. of the 19th Conference of the International Society for Music Information Retrieval (ISMIR’18), Sept. 2018, Paris, France.

annotations: tempo (bpm), genre

### **notes:**

The audio files (664 files, size ~1gb) can be downloaded from <http://www.beatport.com/> using the bash script:

[https://github.com/GiantSteps/giantsteps-tempo-dataset/blob/master/audio\\_dl.sh](https://github.com/GiantSteps/giantsteps-tempo-dataset/blob/master/audio_dl.sh)

To download the files manually use links of the following form: <http://geo-samples.beatport.com/lofi/<name of mp3 file>> e.g.: <http://geo-samples.beatport.com/lofi/5377710.LOFI.mp3>

To convert the audio files to .wav use (bash + sox):



```
./convert_audio.sh
```

To retrieve the genre information, the JSON contained within the website was parsed. The tempo annotation was extracted from forum entries of people correcting the bpm values (i.e. manual annotation of tempo). For more information please contact creators.

[2] found some files without tempo. There are:

```
3041381.LOFI.mp3 3041383.LOFI.mp3 1327052.LOFI.mp3
```

Their v2 tempo is denoted as 0.0 in tempo and mirex and has no annotation in the JAMS format.

(1): Most of the audio files are 120 seconds long. Exceptions are: name length 906760.LOFI.mp3 62 1327052.LOFI.mp3 70 4416506.LOFI.mp3 80 1855660.LOFI.mp3 119 3419452.LOFI.mp3 119 3577631.LOFI.mp3 119

```
class mirdata.giantsteps_tempo.Track(track_id, data_home=None)
    giantsteps_tempo track class :param track_id: track id of the track :type track_id: str :param data_home: Local
    path where the dataset is stored.
```

If *None*, looks for the data in the default directory, *~/mir\_datasets*

```
    audio_path
```

track audio path

**Type** str

```
    title
```

title of the track

**Type** str

```
    track_id
```

track id

**Type** str

```
    annotation_v1_path
```

track annotation v1 path

**Type** str

```
    annotation_v2_path
```

track annotation v2 path

**Type** str

```
    audio
```

audio signal, sample rate

**Type** (np.ndarray, float)

```
    genre
```

human-labeled metadata annotation

**Type** genre

```
    tempo
```

tempo annotation ordered by confidence

**Type** TempoData

**tempo\_v2**  
 tempos annotation ordered by confidence

**Type** TempoData

**to\_jams()**  
 Jams: the track's data in jams format

**to\_jams\_v2()**  
 Jams: the track's data in jams format

**mirdata.giantsteps\_tempo.cite()**  
 Print the reference

**mirdata.giantsteps\_tempo.download**(*data\_home=None, force\_overwrite=False, cleanup=True*)  
 Download the giantsteps\_tempo Dataset (annotations). The audio files are not provided due to copyright issues.  
 :param data\_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, *~/mir\_datasets*

#### Parameters

- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.
- **partial\_download** (*list of str*) – arguments can be 'audio' 'metadata' or/and 'tempos'

**mirdata.giantsteps\_tempo.load**(*data\_home=None*)  
 Load giantsteps\_tempo dataset :param data\_home: Local path where the dataset is stored.  
 If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

**mirdata.giantsteps\_tempo.load\_audio**(*audio\_path*)  
 Load a giantsteps\_tempo audio file. :param audio\_path: path to audio file :type audio\_path: str  
**Returns** the mono audio signal sr (float): The sample rate of the audio file  
**Return type** y (np.ndarray)

**mirdata.giantsteps\_tempo.load\_genre**(*path*)  
 Load genre data from a file :param path: path to metadata annotation file :type path: str  
**Returns** loaded genre data  
**Return type** (str)

**mirdata.giantsteps\_tempo.load\_tempo**(*tempo\_path*)  
 Load giantsteps\_tempo tempo data from a file ordered by confidence :param tempo\_path: path to tempo annotation file :type tempo\_path: str  
**Returns** loaded tempo data  
**Return type** (list of utils.TempoData)

**mirdata.giantsteps\_tempo.track\_ids**()  
 Get the list of track IDs for this dataset :returns: A list of track ids :rtype: (list)

`mirdata.giantsteps_tempo.validate` (*data\_home=None, silence=False*)

Validate if a local version of this dataset is consistent :param data\_home: Local path where the dataset is stored.

If *None*, looks for the data in the default directory, `~/mir_datasets`

#### Returns

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths where the expected file exists locally but has a different checksum than the reference

**Return type** missing\_files (list)

### mirdata.giantsteps\_key

giantsteps\_key Dataset Loader

The GiantSteps+ EDM Key Dataset includes 600 two-minute sound excerpts from various EDM subgenres, annotated with single-key labels, comments and confidence levels by Daniel G. Camhi, and thoroughly revised and expanded by Ángel Faraldo at MTG UPF. Additionally, 500 tracks have been thoroughly analysed, containing pitch-class set descriptions, key changes, and additional modal changes. This dataset is a revision of the original GiantSteps Key Dataset, available in Github (<<https://github.com/GiantSteps/giantsteps-key-dataset>>) and initially described in:

Knees, P., Faraldo, Á., Herrera, P., Vogl, R., Böck, S., Hörschläger, F., Le Goff, M. (2015). Two Datasets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections. In Proceedings of the 16th International Society for Music Information Retrieval Conference, 364–370. Málaga, Spain.

The original audio samples belong to online audio snippets from Beatport, an online music store for DJ's and Electronic Dance Music Producers (<<http://www.beatport.com>>). If this dataset were used in further research, we would appreciate the citation of the current DOI (10.5281/zenodo.1101082) and the following doctoral dissertation, where a detailed description of the properties of this dataset can be found:

Ángel Faraldo (2017). Tonality Estimation in Electronic Dance Music: A Computational and Musically Informed Examination. PhD Thesis. Universitat Pompeu Fabra, Barcelona.

This dataset is mainly intended to assess the performance of computational key estimation algorithms in electronic dance music subgenres.

All the data of this dataset is licensed with Creative Commons Attribution Share Alike 4.0 International.

**class** `mirdata.giantsteps_key.Track` (*track\_id, data\_home=None*)  
giantsteps\_key track class

#### Parameters

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**audio\_path**

track audio path

**Type** str

**keys\_path**

key annotation path

**Type** str

**metadata\_path**

sections annotation path

```

        Type str
title
    title of the track
        Type str
track_id
    track id
        Type str
artists
    artist annotation
        Type Dict
audio
    audio signal, sample rate
        Type (np.ndarray, float)
genres
    genre annotation
        Type Dict
key
    key annotation
        Type String
tempo
    tempo beatports crowdsourced annotation
        Type int
to_jams()
    Jams: the track's data in jams format
mirdata.giantsteps_key.cite()
    Print the reference
mirdata.giantsteps_key.download(data_home=None, force_overwrite=False, cleanup=True, partial_download=None)
    Download the giantsteps_key Dataset (annotations). The audio files are not provided due to copyright issues.

Parameters
    • data_home (str) – Local path where the dataset is stored. If None, looks for the data in the default directory, ~/mir_datasets
    • force_overwrite (bool) – Whether to overwrite the existing downloaded data
    • cleanup (bool) – Whether to delete the zip/tar file after extracting.
    • partial_download (list of str) – arguments can be ‘audio’ ‘metadata’ or/and ‘keys’

mirdata.giantsteps_key.load(data_home=None)
    Load giantsteps_key dataset

Parameters data_home (str) – Local path where the dataset is stored. If None, looks for the data in the default directory, ~/mir_datasets

Returns {track_id: track data}

```

**Return type** (dict)

`mirdata.giantsteps_key.load_artist(metadata_path)`

Load giantsteps\_key tempo data from a file

**Parameters** `metadata_path` (*str*) – path to metadata annotation file

**Returns** list of artists involved in the track.

**Return type** (list of strings)

`mirdata.giantsteps_key.load_audio(audio_path)`

Load a giantsteps\_key audio file.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.giantsteps_key.load_genre(metadata_path)`

Load giantsteps\_key genre data from a file

**Parameters** `metadata_path` (*str*) – path to metadata annotation file

**Returns** with the list of strings with genres ['genres'] and list of strings with sub-genres ['sub-genres']

**Return type** (dict)

`mirdata.giantsteps_key.load_key(keys_path)`

Load giantsteps\_key format key data from a file

**Parameters** `keys_path` (*str*) – path to key annotation file

**Returns** loaded key data

**Return type** (*str*)

`mirdata.giantsteps_key.load_tempo(metadata_path)`

Load giantsteps\_key tempo data from a file

**Parameters** `metadata_path` (*str*) – path to metadata annotation file

**Returns** loaded tempo data

**Return type** (*str*)

`mirdata.giantsteps_key.track_ids()`

Get the list of track IDs for this dataset

**Returns** A list of track ids

**Return type** (list)

`mirdata.giantsteps_key.validate(data_home=None, silence=False)`

Validate if a local version of this dataset is consistent

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths where the expected file exists locally but has a different checksum than the reference

**Return type** missing\_files (list)

## mirdata.groove\_midi

### Groove MIDI Loader

The Groove MIDI Dataset (GMD) is composed of 13.6 hours of aligned MIDI and synthesized audio of human-performed, tempo-aligned expressive drumming. The dataset contains 1,150 MIDI files and over 22,000 measures of drumming.

To enable a wide range of experiments and encourage comparisons between methods on the same data, Gillick et al. created a new dataset of drum performances recorded in MIDI format. They hired professional drummers and asked them to perform in multiple styles to a click track on a Roland TD-11 electronic drum kit. They also recorded the aligned, high-quality synthesized audio from the TD-11 and include it in the release.

The Groove MIDI Dataset (GMD), has several attributes that distinguish it from existing ones:

- The dataset contains about 13.6 hours, 1,150 MIDI files, and over 22,000 measures of drumming.
- Each performance was played along with a metronome set at a specific tempo by the drummer.
- The data includes performances by a total of 10 drummers, with more than 80% of duration coming from hired professionals. The professionals were able to improvise in a wide range of styles, resulting in a diverse dataset.
- The drummers were instructed to play a mix of long sequences (several minutes of continuous playing) and short beats and fills.
- Each performance is annotated with a genre (provided by the drummer), tempo, and anonymized drummer ID.
- Most of the performances are in 4/4 time, with a few examples from other time signatures.
- Four drummers were asked to record the same set of 10 beats in their own style. These are included in the test set split, labeled eval-session/groove1-10.
- In addition to the MIDI recordings that are the primary source of data for the experiments in this work, the authors captured the synthesized audio outputs of the drum set and aligned them to within 2ms of the corresponding MIDI files.

A train/validation/test split configuration is provided for easier comparison of model accuracy on various tasks.

The dataset is made available by Google LLC under a Creative Commons Attribution 4.0 International (CC BY 4.0) License.

For more details, please visit: <http://magenta.tensorflow.org/datasets/groove>

**class** mirdata.groove\_midi.**Track** (*track\_id*, *data\_home=None*)  
Groove MIDI Track class

#### Parameters

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

#### drummer

Drummer id of the track (ex. 'drummer1')

**Type** str

#### session

Type of session (ex. 'session1', 'eval\_session')

**Type** str

**track\_id**  
track id of the track (ex. 'drummer1/eval\_session/1')

**Type** str

**style**  
Style (genre, groove type) of the track (ex. 'funk/groove1')

**Type** str

**tempo**  
Track tempo in beats per minute (ex. 138)

**Type** int

**beat\_type**  
Whether the track is a beat or a fill (ex. 'beat')

**Type** str

**time\_signature**  
Time signature of the track (ex. '4-4', '6-8')

**Type** str

**midi\_path**  
Path to the midi file

**Type** str

**audio\_path**  
Path to the audio file

**Type** str

**duration**  
Duration of the midi file in seconds

**Type** float

**split**  
Whether the track is for a train/valid/test set. One of 'train', 'valid' or 'test'.

**Type** str

**audio**  
audio signal, sample rate

**Type** (np.ndarray, float)

**beats**  
machine-generated beat annotation

**Type** BeatData

**drum\_events**  
annotated drum kit events

**Type** EventData

**midi**  
prettyMIDI obj

**Type** (obj)

`mirdata.groove_midi.cite()`  
Print the reference

`mirdata.groove_midi.download(data_home=None, force_overwrite=False, cleanup=True)`

Download Groove MIDI.

**Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.groove_midi.load(data_home=None)`

Load Groove MIDI dataset

**Parameters** **data\_home** (*str*) – Local path where Groove MIDI is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.groove_midi.load_audio(audio_path)`

Load a Groove MIDI audio file.

**Parameters** **audio\_path** (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.groove_midi.load_beats(midi_path, midi=None)`

Load beat data from the midi file.

**Parameters**

- **midi\_path** (*str*) – path to midi file
- **midi** (*pretty\_midi.PrettyMIDI*) – pre-loaded midi object or *None* if *None*, the midi object is loaded using *midi\_path*

**Returns** *beat\_data* (BeatData)

`mirdata.groove_midi.load_drum_events(midi_path, midi=None)`

Load drum events from the midi file.

**Parameters**

- **midi\_path** (*str*) – path to midi file
- **midi** (*pretty\_midi.PrettyMIDI*) – pre-loaded midi object or *None* if *None*, the midi object is loaded using *midi\_path*

**Returns** *drum\_events* (EventData)

`mirdata.groove_midi.load_midi(midi_path)`

Load a Groove MIDI midi file.

**Parameters** **midi\_path** (*str*) – path to midi file

**Returns** *pretty\_midi* object

**Return type** *midi\_data* (*pretty\_midi.PrettyMIDI*)

`mirdata.groove_midi.track_ids()`

Return track ids

**Returns** A list of track ids



**Return type** (list)

`mirdata.groove_midi.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list): List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (list)

## mirdata.gtzan\_genre

GTZAN-Genre Dataset Loader

This dataset was used for the well known paper in genre classification “Musical genre classification of audio signals ” by G. Tzanetakis and P. Cook in IEEE Transactions on Audio and Speech Processing 2002.

The dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres, each represented by 100 tracks. The tracks are all 22050 Hz mono 16-bit audio files in .wav format.

**class** `mirdata.gtzan_genre.Track` (*track\_id*, *data\_home=None*)  
gtzan\_genre Track class

**Parameters**

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, `~/mir_datasets`

**audio\_path**

path to the audio file

**Type** `str`

**genre**

annotated genre

**Type** `str`

**track\_id**

track id

**Type** `str`

**audio**

audio signal, sample rate

**Type** (`np.ndarray`, `float`)

**to\_jams()**

Jams: the track’s data in jams format

`mirdata.gtzan_genre.cite()`

Print the reference

`mirdata.gtzan_genre.download(data_home=None, force_overwrite=False, cleanup=True)`

Download the GTZAN-Genre dataset.

### Parameters

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.gtzan_genre.load(data_home=None)`

Load GTZAN-Genre

**Parameters** **data\_home** (*str*) – Local path where GTZAN-Genre is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.gtzan_genre.load_audio(audio_path)`

Load a GTZAN audio file.

**Parameters** **audio\_path** (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.gtzan_genre.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.gtzan_genre.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list): List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** *missing\_files* (list)

## mirdata.guitarset

### GuitarSet Loader

GuitarSet provides audio recordings of a variety of musical excerpts played on an acoustic guitar, along with time-aligned annotations including pitch contours, string and fret positions, chords, beats, downbeats, and keys.

GuitarSet contains 360 excerpts that are close to 30 seconds in length. The 360 excerpts are the result of the following combinations:

- 6 players
- 2 versions: comping (harmonic accompaniment) and soloing (melodic improvisation)
- 5 styles: Rock, Singer-Songwriter, Bossa Nova, Jazz, and Funk

- 3 Progressions: 12 Bar Blues, Autumn Leaves, and Pachelbel Canon.
- 2 Tempi: slow and fast.

The tonality (key) of each excerpt is sampled uniformly at random.

GuitarSet was recorded with the help of a hexaphonic pickup, which outputs signals for each string separately, allowing automated note-level annotation. Excerpts are recorded with both the hexaphonic pickup and a Neumann U-87 condenser microphone as reference. 3 audio recordings are provided with each excerpt with the following suffix:

- hex: original 6 channel wave file from hexaphonic pickup
- hex\_cln: hex wave files with interference removal applied
- mic: monophonic recording from reference microphone
- mix: monophonic mixture of original 6 channel file

Each of the 360 excerpts has an accompanying JAMS file which stores 16 annotations. Pitch:

- 6 pitch\_contour annotations (1 per string)
- 6 midi\_note annotations (1 per string)

Beat and Tempo:

- 1 beat\_position annotation
- 1 tempo annotation

Chords:

- 2 chord annotations: instructed and performed. The instructed chord annotation

is a digital version of the lead sheet that's provided to the player, and the performed chord annotations are inferred from note annotations, using segmentation and root from the digital lead sheet annotation.

For more details, please visit: <http://github.com/marl/guitarset/>

```
class mirdata.guitarset.Track (track_id, data_home=None)
    guitarset Track class
```

#### Parameters

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

**audio\_hex\_cln\_path**

path to the debleeded hex wave file

**Type** str

**audio\_hex\_path**

path to the original hex wave file

**Type** str

**audio\_mic\_path**

path to the mono wave via microphone

**Type** str

**audio\_mix\_path**

path to the mono wave via downmixing hex pickup

**Type** str

**jams\_path**

path to the jams file

**Type** str

**mode**

one of ['solo', 'comp'] For each excerpt, players are asked to first play in 'comp' mode and later play a 'solo' version on top of the already recorded comp.

**Type** str

**player\_id**

ID of the different players. one of ['00', '01', ..., '05']

**Type** str

**style**

one of ['Jazz', 'Bossa Nova', 'Rock', 'Singer-Songwriter', 'Funk']

**Type** str

**tempo**

BPM of the track

**Type** float

**track\_id**

track id

**Type** str

**audio\_hex**

raw hexaphonic audio signal, sample rate

**Type** (np.ndarray, float)

**audio\_hex\_cln**

bleed-removed hexaphonic audio signal, sample rate

**Type** (np.ndarray, float)

**audio\_mic**

stereo microphone audio signal, sample rate

**Type** (np.ndarray, float)

**audio\_mix**

stereo mix audio signal, sample rate

**Type** (np.ndarray, float)

**beats**

the track's beat positions

**Type** BeatData

**inferred\_chords**

the track's chords inferred from played transcription

**Type** ChordData

**key\_mode**

the track's key and mode

**Type** KeyData

**leadsheet\_chords**

the track's chords as written in the leadsheet

**Type** ChordData

**notes**

a dict that contains 6 NoteData. From Low E string to high e string. {

    'E': NoteData(...), 'A': NoteData(...), ... 'e': NoteData(...)

}

**Type** dict

**pitch\_contours**

a dict that contains 6 F0Data. From Low E string to high e string. {

    'E': F0Data(...), 'A': F0Data(...), ... 'e': F0Data(...)

}

**Type** (dict)

**to\_jams()**

Jams: the track's data in jams format

`mirdata.guitarset.cite()`

Print the reference

`mirdata.guitarset.download(data_home=None, partial_download=None, force_overwrite=False, cleanup=True)`

Download GuitarSet.

**Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial\_download** (*list*) –

**List indicating what to partially download. The list can include any of:**

- `'annotations'` the annotation files
- `'audio_hex_original'` original 6 channel wave file from hexaphonic pickup
- `'audio_hex_debleded'` hex wave files with interference removal applied
- `'audio_mic'` monophonic recording from reference microphone
- `'audio_mix'` monophonic mixture of original 6 channel file

If *None*, all data is downloaded.

- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.guitarset.load(data_home=None)`

Load GuitarSet

**Parameters** **data\_home** (*str*) – Local path where GuitarSet is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.guitarset.load_audio(audio_path)`

Load a Guitarset audio file.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.guitarset.load_chords(jams_path, leadsheet_version=True)`

**Parameters**

- `jams_path` (*str*) – Path of the jams annotation file
- `leadsheet_version` (*Bool*) – Whether or not to load the leadsheet version of the chord annotation. If False, load the inferred version.

**Returns** Chord data

**Return type** (ChordData)

`mirdata.guitarset.load_multitrack_audio(audio_path)`

Load a Guitarset multitrack audio file.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.guitarset.load_note_ann(jams_path, string_num)`

**Parameters** `jams_path` (*str*) – Path of the jams annotation file

**string\_num** (int), in range(6): Which string to load. 0 is the Low E string, 5 is the high e string.

`mirdata.guitarset.load_pitch_contour(jams_path, string_num)`

**Parameters** `jams_path` (*str*) – Path of the jams annotation file

**string\_num** (int), in range(6): Which string to load. 0 is the Low E string, 5 is the high e string.

`mirdata.guitarset.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.guitarset.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums** (list): **List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (list)

## mirdata.ikala

### iKala Dataset Loader

The iKala dataset is comprised of 252 30-second excerpts sampled from 206 iKala songs (plus 100 hidden excerpts reserved for MIREX). The music accompaniment and the singing voice are recorded at the left and right channels respectively and can be found under the Wavfile directory. In addition, the human-labeled pitch contours and times-tamped lyrics can be found under PitchLabel and Lyrics respectively.

For more details, please visit: <http://mac.citi.sinica.edu.tw/ikala/>

**class** mirdata.ikala.**Track** (*track\_id*, *data\_home=None*)  
 ikala Track class

#### Parameters

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

#### **audio\_path**

path to the track's audio file

**Type** *str*

#### **f0\_path**

path to the track's f0 annotation file

**Type** *str*

#### **lyrics\_path**

path to the track's lyric annotation file

**Type** *str*

#### **section**

section. Either 'verse' or 'chorus'

**Type** *str*

#### **singer\_id**

singer id

**Type** *str*

#### **song\_id**

song id of the track

**Type** *str*

#### **track\_id**

track id

**Type** *str*

#### **f0**

The human-annotated singing voice pitch

**Type** *F0Data*

#### **instrumental\_audio**

mono instrumental audio signal, sample rate

**Type** (*np.ndarray*, *float*)

**lyrics**

The human-annotated lyrics

**Type** LyricData

**mix\_audio**

mono mixture audio signal, sample rate

**Type** (np.ndarray, float)

**to\_jams()**

Jams: the track's data in jams format

**vocal\_audio**

mono vocal audio signal, sample rate

**Type** (np.ndarray, float)

`mirdata.ikala.cite()`

Print the reference

`mirdata.ikala.download(data_home=None, force_overwrite=False)`

Download iKala Dataset. However, iKala dataset is not available for download anymore. This function prints a helper message to organize pre-downloaded iKala dataset.

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

`mirdata.ikala.load(data_home=None)`

Load iKala dataset

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns** `{track_id: track data}`

**Return type** (dict)

`mirdata.ikala.load_instrumental_audio(audio_path)`

Load an ikala instrumental.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal sr (float): The sample rate of the audio file

**Return type** `y` (np.ndarray)

`mirdata.ikala.load_mix_audio(audio_path)`

Load an ikala mix.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal sr (float): The sample rate of the audio file

**Return type** `y` (np.ndarray)

`mirdata.ikala.load_vocal_audio(audio_path)`

Load an ikala vocal.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal sr (float): The sample rate of the audio file

**Return type** `y` (np.ndarray)

`mirdata.ikala.track_ids()`

Return track ids



**Returns** A list of track ids

**Return type** (list)

`mirdata.ikala.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list): List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (list)

## mirdata.maestro

### MAESTRO Dataset Loader

MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) is a dataset composed of over 200 hours of virtuosic piano performances captured with fine alignment (~3 ms) between note labels and audio waveforms.

The dataset is created and released by Google’s Magenta team.

The dataset contains over 200 hours of paired audio and MIDI recordings from ten years of International Piano-e-Competition. The MIDI data includes key strike velocities and sustain/sostenuto/una corda pedal positions. Audio and MIDI files are aligned with 3 ms accuracy and sliced to individual musical pieces, which are annotated with composer, title, and year of performance. Uncompressed audio is of CD quality or higher (44.1–48 kHz 16-bit PCM stereo).

A train/validation/test split configuration is also proposed, so that the same composition, even if performed by multiple contestants, does not appear in multiple subsets. Repertoire is mostly classical, including composers from the 17th to early 20th century.

The dataset is made available by Google LLC under a Creative Commons Attribution Non-Commercial Share-Alike 4.0 (CC BY-NC-SA 4.0) license.

This loader supports MAESTRO version 2.

For more details, please visit: <https://magenta.tensorflow.org/datasets/maestro>

**class** `mirdata.maestro.Track` (*track\_id*, *data\_home=None*)

MAESTRO Track class

**Parameters**

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, `~/mir_datasets`

**audio\_path**

Path to the track’s audio file

**Type** `str`

**canonical\_composer**

Composer of the piece, standardized on a single spelling for a given name.

**Type** `str`

**canonical\_title**

Title of the piece. Not guaranteed to be standardized to a single representation.

**Type** str

**duration**

Duration in seconds, based on the MIDI file.

**Type** float

**midi\_path**

Path to the track's MIDI file

**Type** str

**split**

Suggested train/validation/test split.

**Type** str

**track\_id**

track id

**Type** str

**year**

Year of performance.

**Type** int

**audio**

track's audio signal, sample rate

**Type** (np.ndarray, float)

**midi**

description of output

**Type** output type

**notes**

annotated piano notes

**Type** NoteData

**to\_jams()**

Jams: the track's data in jams format

**mirdata.maestro.cite()**

Print the reference

**mirdata.maestro.download**(*data\_home=None*, *partial\_download=None*, *force\_overwrite=False*, *cleanup=True*)

Download the dataset.

**Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial\_download** (*list*) –  
 List indicating what to partially download. The list can include any of:
  - 'all': audio, midi and metadata

- 'midi': midi and metadata only
- 'metadata': metadata only

If *None*, all data is downloaded.

- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.maestro.load(data_home=None)`

Load MAESTRO dataset

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.maestro.load_audio(audio_path)`

Load a MAESTRO audio file.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.maestro.load_midi(midi_path)`

Load a MAESTRO midi file.

**Parameters** `midi_path` (*str*) – path to midi file

**Returns** *pretty\_midi* object

**Return type** *midi\_data* (obj)

`mirdata.maestro.load_notes(midi_path, midi=None)`

Load note data from the midi file.

**Parameters**

- **midi\_path** (*str*) – path to midi file
- **midi** (*pretty\_midi.PrettyMIDI*) – pre-loaded midi object or *None* if *None*, the midi object is loaded using *midi\_path*

**Returns** *note\_data* (NoteData)

`mirdata.maestro.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.maestro.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

**Return type** missing\_files (list)

## **mirdata.medleydb\_melody**

MedleyDB melody Dataset Loader

MedleyDB is a dataset of annotated, royalty-free multitrack recordings. MedleyDB was curated primarily to support research on melody extraction, addressing important shortcomings of existing collections. For each song we provide melody f0 annotations as well as instrument activations for evaluating automatic instrument recognition.

For more details, please visit: <https://medleydb.weebly.com>

**class** mirdata.medleydb\_melody.**Track** (*track\_id*, *data\_home=None*)  
medleydb\_melody Track class

### **Parameters**

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

**artist**

artist

**Type** str

**audio\_path**

path to the audio file

**Type** str

**genre**

genre

**Type** str

**is\_excerpt**

True if the track is an excerpt

**Type** bool

**is\_instrumental**

True if the track does not contain vocals

**Type** bool

**melody1\_path**

path to the melody1 annotation file

**Type** str

**melody2\_path**

path to the melody2 annotation file

**Type** str

**melody3\_path**

path to the melody3 annotation file

**Type** str

**n\_sources**

Number of instruments in the track

**Type** int

**title**  
title

**Type** str

**track\_id**  
track id

**Type** str

**audio**  
audio signal, sample rate

**Type** (np.ndarray, float)

**melody1**  
The pitch of the single most predominant source (often the voice)

**Type** F0Data

**melody2**  
The pitch of the predominant source for each point in time

**Type** F0Data

**melody3**  
The pitch of any melodic source. Allows for more than one f0 value at a time.

**Type** *MultipitchData*

**to\_jams()**  
Jams: the track's data in jams format

`mirdata.medleydb_melody.cite()`  
Print the reference

`mirdata.medleydb_melody.download(data_home=None)`  
MedleyDB is not available for downloading directly. This function prints a helper message to download MedleyDB through zenodo.org.

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

`mirdata.medleydb_melody.load(data_home=None)`  
Load MedleyDB melody dataset

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.medleydb_melody.load_audio(audio_path)`  
Load a MedleyDB audio file.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.medleydb_melody.track_ids()`  
Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.medleydb_melody.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list): List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (list)

## mirdata.medleydb\_pitch

MedleyDB pitch Dataset Loader

MedleyDB is a dataset of annotated, royalty-free multitrack recordings. MedleyDB was curated primarily to support research on melody extraction, addressing important shortcomings of existing collections. For each song we provide melody f0 annotations as well as instrument activations for evaluating automatic instrument recognition.

For more details, please visit: <https://medleydb.weebly.com>

**class** `mirdata.medleydb_pitch.Track(track_id, data_home=None)`

`medleydb_pitch` Track class

**Parameters**

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, `~/mir_datasets`

**artist**

artist

**Type** *str*

**audio\_path**

path to the audio file

**Type** *str*

**genre**

genre

**Type** *str*

**instrument**

instrument of the track

**Type** *str*

**pitch\_path**

path to the pitch annotation file

**Type** *str*

**title**  
title  
**Type** str

**track\_id**  
track id  
**Type** str

**audio**  
audio signal, sample rate  
**Type** (np.ndarray, float)

**pitch**  
The human-annotated pitch  
**Type** F0Data

**to\_jams()**  
Jams: the track's data in jams format

`mirdata.medleydb_pitch.cite()`  
Print the reference

`mirdata.medleydb_pitch.download(data_home=None)`  
MedleyDB is not available for downloading directly. This function prints a helper message to download MedleyDB through zenodo.org.  
**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

`mirdata.medleydb_pitch.load(data_home=None)`  
Load MedleyDB pitch dataset  
**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`  
**Returns** {*track\_id*: track data}  
**Return type** (dict)

`mirdata.medleydb_pitch.load_audio(audio_path)`  
Load a MedleyDB audio file.  
**Parameters** `audio_path` (*str*) – path to audio file  
**Returns** the mono audio signal sr (float): The sample rate of the audio file  
**Return type** y (np.ndarray)

`mirdata.medleydb_pitch.track_ids()`  
Return track ids  
**Returns** A list of track ids  
**Return type** (list)

`mirdata.medleydb_pitch.validate(data_home=None, silence=False)`  
Validate if the stored dataset is a valid version  
**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`  
**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

**Return type** missing\_files (list)

## mirdata.medley\_solos\_db

Medley-solos-DB Dataset Loader.

Medley-solos-DB is a cross-collection dataset for automatic musical instrument recognition in solo recordings. It consists of a training set of 3-second audio clips, which are extracted from the MedleyDB dataset (Bittner et al., ISMIR 2014) as well as a test set of 3-second clips, which are extracted from the solosDB dataset (Essid et al., IEEE TASLP 2009). Each of these clips contains a single instrument among a taxonomy of eight:

0. clarinet,
1. distorted electric guitar,
2. female singer,
3. flute,
4. piano,
5. tenor saxophone,
6. trumpet, and
7. violin.

The Medley-solos-DB dataset is the dataset that is used in the benchmarks of musical instrument recognition in the publications of Lostanlen and Cella (ISMIR 2016) and Andén et al. (IEEE TSP 2019).

**class** mirdata.medley\_solos\_db.**Track** (*track\_id*, *data\_home=None*)  
 medley\_solos\_db Track class

### Parameters

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

### audio\_path

path to the track's audio file

**Type** str

### instrument

instrument encoded by its English name

**Type** str

### instrument\_id

instrument encoded as an integer

**Type** int

### song\_id

song encoded as an integer

**Type** int



**subset**

either equal to 'train', 'validation', or 'test'

**Type** str

**track\_id**

track id

**Type** str

**audio**

audio signal, sample rate

**Type** (np.ndarray, float)

**to\_jams()**

Jams: the track's data in jams format

`mirdata.medley_solos_db.cite()`

Print the reference

`mirdata.medley_solos_db.download(data_home=None, partial_download=None, force_overwrite=False, cleanup=True)`

Download Medley-solos-DB.

**Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial\_download** (*list*) –

**List indicating what to partially download. The list can include any of:**

- *'annotations'* the annotation files
- *'audio'* the audio files

If *None*, all data is downloaded.

- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.medley_solos_db.load(data_home=None)`

Load Medley-solos-DB :param data\_home: Local path where Medley-solos-DB is stored.

If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.medley_solos_db.load_audio(audio_path)`

Load a Medley Solos DB audio file.

**Parameters** **audio\_path** (*str*) – path to audio file

**Returns** the mono audio signal sr (float): The sample rate of the audio file

**Return type** y (np.ndarray)

`mirdata.medley_solos_db.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.medley_solos_db.validate` (*data\_home=None, silence=False*)

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list): List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (list)

## mirdata.orchset

ORCHSET Dataset Loader

Orchset is intended to be used as a dataset for the development and evaluation of melody extraction algorithms. This collection contains 64 audio excerpts focused on symphonic music with their corresponding annotation of the melody.

For more details, please visit: <https://zenodo.org/record/1289786#.XREpzaeZPx6>

**class** `mirdata.orchset.Track` (*track\_id, data\_home=None*)

orchset Track class

**Parameters**

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, `~/mir_datasets`

**alternating\_melody**

True if the melody alternates between instruments

**Type** bool

**audio\_path\_mono**

path to the mono audio file

**Type** str

**audio\_path\_stereo**

path to the stereo audio file

**Type** str

**composer**

the work's composer

**Type** str

**contains\_brass**

True if the track contains any brass instrument

**Type** bool

**contains\_strings**

True if the track contains any string instrument

**Type** bool

**contains\_winds**

True if the track contains any wind instrument

**Type** bool

**excerpt**

True if the track is an excerpt

**Type** str

**melody\_path**

path to the melody annotation file

**Type** str

**only\_brass**

True if the track contains brass instruments only

**Type** bool

**only\_strings**

True if the track contains string instruments only

**Type** bool

**only\_winds**

True if the track contains wind instruments only

**Type** bool

**predominant\_melodic\_instruments**

List of instruments which play the melody

**Type** list

**track\_id**

track id

**Type** str

**work**

The musical work

**Type** str

**audio\_mono**

mono audio signal, sample rate

**Type** (np.ndarray, float)

**audio\_stereo**

stereo audio signal, sample rate

**Type** (np.ndarray, float)

**melody**

melody annotation

**Type** F0Data

**to\_jams()**

Jams: the track's data in jams format

**mirdata.orchset.cite()**

Print the reference

`mirdata.orchset.download(data_home=None, force_overwrite=False, cleanup=True)`

Download ORCHSET Dataset.

**Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.orchset.load(data_home=None)`

Load ORCHSET dataset

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.orchset.load_audio_mono(audio_path)`

Load a Orchset audio file.

**Parameters** **audio\_path** (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.orchset.load_audio_stereo(audio_path)`

Load a Orchset audio file.

**Parameters** **audio\_path** (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.orchset.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.orchset.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters**

- **dataset\_path** (*str*) – ORCHSET dataset local path
- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

**Return type** *missing\_files* (list)

**mirdata.rwc\_classical**

## RWC Classical Dataset Loader

The Classical Music Database consists of 50 pieces:

- Symphonies: 4 pieces
- Concerti: 2 pieces
- Orchestral music: 4 pieces
- Chamber music: 10 pieces
- Solo performances: 24 pieces
- Vocal performances: 6 pieces

For more details, please visit: <https://staff.aist.go.jp/m.goto/RWC-MDB/rwc-mdb-c.html>

**class** mirdata.rwc\_classical.Track(track\_id, data\_home=None)  
 rwc\_classical Track class

**Parameters**

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

**artist**

the track's artist

**Type** str

**audio\_path**

path of the audio file

**Type** str

**beats\_path**

path of the beat annotation file

**Type** str

**category**

One of 'Symphony', 'Concerto', 'Orchestral', 'Solo', 'Chamber', 'Vocal', or blank.

**Type** str

**composer**

Composer of this Track.

**Type** str

**duration**

Duration of the track in seconds

**Type** float

**piece\_number**

Piece number of this Track, [1-50]

**Type** str

**sections\_path**

path of the section annotation file

**Type** str

**suffix**  
string within M01-M06

**Type** str

**title**  
Title of The track.

**Type** str

**track\_id**  
track id

**Type** str

**track\_number**  
CD track number of this Track

**Type** str

**audio**  
audio signal, sample rate

**Type** (np.ndarray, float)

**beats**  
human labeled beat annotations

**Type** BeatData

**sections**  
human labeled section annotations

**Type** SectionData

**to\_jams()**  
Jams: the track's data in jams format

`mirdata.rwc_classical.download(data_home=None, partial_download=None, force_overwrite=False, cleanup=True)`  
Download the RWC Classical (annotations and metadata). The audio files are not provided due to copyright issues.

#### Parameters

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial\_download** (*list*) – List indicating what to partially download. The list can include any of \* `'annotations_beat'` the beat annotation files \* `'annotations_sections'` the sections annotation files \* `'metadata'` the metadata files If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.rwc_classical.load(data_home=None)`  
Load RWC-Classical dataset

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.rwc_classical.load_audio(audio_path)`

Load a RWC audio file.

**Parameters** `audio_path` (*str*) – path to audio file

**Returns** the mono audio signal *sr* (float): The sample rate of the audio file

**Return type** *y* (np.ndarray)

`mirdata.rwc_classical.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.rwc_classical.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list): List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (list)

## mirdata.rwc\_jazz

RWC Jazz Dataset Loader.

The Jazz Music Database consists of 50 pieces:

- Instrumentation variations: 35 pieces (5 pieces × 7 instrumentations).

The instrumentation-variation pieces were recorded to obtain different versions of the same piece; i.e., different arrangements performed by different player instrumentations. Five standard-style jazz pieces were originally composed and then performed in modern-jazz style using the following seven instrumentations: 1. Piano solo 2. Guitar solo 3. Duo: Vibraphone + Piano, Flute + Piano, and Piano + Bass 4. Piano trio: Piano + Bass + Drums 5. Piano trio + Trumpet or Tenor saxophone 6. Octet: Piano trio + Guitar + Alto saxophone + Baritone saxophone + Tenor saxophone × 2 7. Piano trio + Vibraphone or Flute

- Style variations: 9 pieces

The style-variation pieces were recorded to represent various styles of jazz. They include four well-known public-domain pieces and consist of 1. Vocal jazz: 2 pieces (including “Aura Lee”) 2. Big band jazz: 2 pieces (including “The Entertainer”) 3. Modal jazz: 2 pieces 4. Funky jazz: 2 pieces (including “Silent Night”) 5. Free jazz: 1 piece (including “Joyful, Joyful, We Adore Thee”) Fusion (crossover): 6 pieces The fusion pieces were recorded to obtain music that combines elements of jazz with other styles such as popular, rock, and latin. They include music with an eighth-note feel, music with a sixteenth-note feel, and Latin jazz music.

For more details, please visit: <https://staff.aist.go.jp/m.goto/RWC-MDB/rwc-mdb-j.html>

**class** `mirdata.rwc_jazz.Track` (*track\_id*, *data\_home=None*)

`rwc_jazz` Track class

**Parameters**

- **track\_id** (*str*) – track id of the track

- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

**artist**

Artist name

**Type** str

**audio\_path**

path of the audio file

**Type** str

**beats\_path**

path of the beat annotation file

**Type** str

**duration**

Duration of the track in seconds

**Type** float

**instruments**

list of used instruments.

**Type** str

**piece\_number**

Piece number of this Track, [1-50]

**Type** str

**sections\_path**

path of the section annotation file

**Type** str

**suffix**

M01-M04

**Type** str

**title**

Title of The track.

**Type** str

**track\_id**

track id

**Type** str

**track\_number**

CD track number of this Track

**Type** str

**variation**

TODO

**Type** str

**audio**

audio signal, sample rate

**Type** (np.ndarray, float)



### **beats**

human-labeled beat data

**Type** BeatData

### **sections**

human-labeled section data

**Type** SectionData

### **to\_jams()**

Jams: the track's data in jams format

`mirdata.rwc_jazz.download(data_home=None, partial_download=None, force_overwrite=False, cleanup=True)`

Download the RWC Jazz (annotations and metadata). The audio files are not provided due to copyright issues.

#### **Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial\_download** (*list*) – List indicating what to partially download. The list can include any of: \* `'annotations_beat'` the beat annotation files \* `'annotations_sections'` the sections annotation files \* `'metadata'` the metadata files If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.rwc_jazz.load(data_home=None)`

Load RWC-Jazz dataset

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.rwc_jazz.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.rwc_jazz.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

#### **Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

**Return type** missing\_files (list)

## mirdata.rwc\_popular

### RWC Popular Dataset Loader

The Popular Music Database consists of 100 songs — 20 songs with English lyrics performed in the style of popular music typical of songs on the American hit charts in the 1980s, and 80 songs with Japanese lyrics performed in the style of modern Japanese popular music typical of songs on the Japanese hit charts in the 1990s.

For more details, please visit: <https://staff.aist.go.jp/m.goto/RWC-MDB/rwc-mdb-p.html>

```
class mirdata.rwc_popular.Track(track_id, data_home=None)
    rwc_popular Track class
```

#### Parameters

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

**artist**  
artist

**Type** str

**audio\_path**  
path of the audio file

**Type** str

**beats\_path**  
path of the beat annotation file

**Type** str

**chords\_path**  
path of the chord annotation file

**Type** str

**drum\_information**  
If the drum is ‘Drum sequences’, ‘Live drums’, or ‘Drum loops’

**Type** str

**duration**  
Duration of the track in seconds

**Type** float

**instruments**  
List of used instruments

**Type** str

**piece\_number**  
Piece number, [1-50]

**Type** str

**sections\_path**  
path of the section annotation file

**Type** str

**singer\_information**  
 TODO  
**Type** str

**suffix**  
 M01-M04  
**Type** str

**tempo**  
 Tempo of the track in BPM  
**Type** str

**title**  
 title  
**Type** str

**track\_id**  
 track id  
**Type** str

**track\_number**  
 CD track number  
**Type** str

**voca\_inst\_path**  
 path of the vocal/instrumental annotation file  
**Type** str

**audio**  
 audio signal, sample rate  
**Type** (np.ndarray, float)

**beats**  
 human-labeled beat annotation  
**Type** BeatData

**chords**  
 human-labeled chord annotation  
**Type** ChordData

**sections**  
 human-labeled section annotation  
**Type** SectionData

**to\_jams** ()  
 Jams: the track's data in jams format

**vocal\_instrument\_activity**  
 human-labeled vocal/instrument activity  
**Type** EventData

`mirdata.rwc_popular.download(data_home=None, partial_download=None, force_overwrite=False, cleanup=True)`

Download the RWC Popular (annotations and metadata). The audio files are not provided due to copyright issues.

### Parameters

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial\_download** (*list*) – List indicating what to partially download. The list can include any of: \* *'annotations\_beat'* the beat annotation files \* *'annotations\_sections'* the sections annotation files \* *'annotations\_chords'* the chords annotation files \* *'annotations\_vocal\_act'* the vocal activity annotation files \* *'metadata'* the metadata files If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.rwc_popular.load(data_home=None)`

Load RWC-Genre dataset

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.rwc_popular.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.rwc_popular.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list): List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** *missing\_files* (list)

## mirdata.salami

### SALAMI Dataset Loader

The SALAMI dataset contains Structural Annotations of a Large Amount of Music Information: the public portion contains over 2200 annotations of over 1300 unique tracks.

NB: mirdata relies on the **corrected** version of the 2.0 annotations: Details can be found at <https://github.com/bmcfee/salami-data-public/tree/hierarchy-corrections> and <https://github.com/DDMAL/salami-data-public/pull/15>.

For more details, please visit: <https://github.com/DDMAL/salami-data-public>

**class** `mirdata.salami.Track(track_id, data_home=None)`

salami Track class

### Parameters

- **track\_id** (*str*) – track id of the track

- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, *~/mir\_datasets*

**annotator\_1\_id**  
number that identifies annotator 1  
**Type** str

**annotator\_1\_time**  
time that the annotator 1 took to complete the annotation  
**Type** str

**annotator\_2\_id**  
number that identifies annotator 1  
**Type** str

**annotator\_2\_time**  
time that the annotator 1 took to complete the annotation  
**Type** str

**artist**  
song artist  
**Type** str

**audio\_path**  
path to the audio file  
**Type** str

**broad\_genre**  
broad genre of the song  
**Type** str

**duration**  
duration of song in seconds  
**Type** float

**genre**  
genre of the song  
**Type** str

**sections\_annotator1\_lowercase\_path**  
path to annotations in hierarchy level 1 from annotator 1  
**Type** str

**sections\_annotator1\_uppercase\_path**  
path to annotations in hierarchy level 0 from annotator 1  
**Type** str

**sections\_annotator2\_lowercase\_path**  
path to annotations in hierarchy level 1 from annotator 2  
**Type** str

**sections\_annotator2\_uppercase\_path**  
path to annotations in hierarchy level 0 from annotator 2  
**Type** str

**source**

dataset or source of song

**Type** str

**title**

title of the song

**Type** str

**audio**

audio signal, sample rate

**Type** (np.ndarray, float)

**sections\_annotator\_1\_lowercase**

annotations in hierarchy level 1 from annotator 1

**Type** SectionData

**sections\_annotator\_1\_uppercase**

annotations in hierarchy level 0 from annotator 1

**Type** SectionData

**sections\_annotator\_2\_lowercase**

annotations in hierarchy level 1 from annotator 2

**Type** SectionData

**sections\_annotator\_2\_uppercase**

annotations in hierarchy level 0 from annotator 2

**Type** SectionData

**to\_jams()**

Jams: the track's data in jams format

`mirdata.salami.cite()`

Print the reference

`mirdata.salami.download(data_home=None, force_overwrite=False, cleanup=True)`

Download SALAMI Dataset (annotations). The audio files are not provided.

**Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.salami.load(data_home=None)`

Load SALAMI dataset

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

`mirdata.salami.load_audio(audio_path)`

Load a Salami audio file.

**Parameters** **audio\_path** (*str*) – path to audio file

**Returns** the mono audio signal sr (float): The sample rate of the audio file

**Return type** y (np.ndarray)

`mirdata.salami.track_ids()`

Return track ids

**Returns** A list of track ids

**Return type** (list)

`mirdata.salami.validate(data_home=None, silence=False)`

Validate if the stored dataset is a valid version

**Parameters** `data_home` (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, `~/mir_datasets`

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums** (list): **List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** missing\_files (list)

## mirdata.tinysol

TinySOL Dataset Loader.

TinySOL is a dataset of 2913 samples, each containing a single musical note from one of 14 different instruments:

Bass Tuba French Horn Trombone Trumpet in C Accordion Contrabass Violin Viola Violoncello Bassoon  
Clarinet in B-flat Flute Oboe Alto Saxophone

These sounds were originally recorded at Ircam in Paris (France) between 1996 and 1999, as part of a larger project named Studio On Line (SOL). Although SOL contains many combinations of mutes and extended playing techniques, TinySOL purely consists of sounds played in the so-called “ordinary” style, and in absence of mute.

TinySOL can be used for education and research purposes. In particular, it can be employed as a dataset for training and/or evaluating music information retrieval (MIR) systems, for tasks such as instrument recognition or fundamental frequency estimation. For this purpose, we provide an official 5-fold split of TinySOL as a metadata attribute. This split has been carefully balanced in terms of instrumentation, pitch range, and dynamics. For the sake of research reproducibility, we encourage users of TinySOL to adopt this split and report their results in terms of average performance across folds.

We encourage TinySOL users to subscribe to the Ircam Forum so that they can have access to larger versions of SOL.

For more details, please visit: <https://www.orch-idea.org/>

**class** `mirdata.tinysol.Track` (*track\_id*, *data\_home=None*)

tinysol Track class

**Parameters**

- **track\_id** (*str*) – track id of the track
- **data\_home** (*str*) – Local path where the dataset is stored. default=None If *None*, looks for the data in the default directory, `~/mir_datasets`

**audio\_path**

path of the audio file

**Type** str

**dynamics**

dynamics abbreviation. Ex: pp, mf, ff, etc.

**Type** str

**dynamics\_id**

pp=0, p=1, mf=2, f=3, ff=4

**Type** int

**family**

instrument family encoded by its English name

**Type** str

**instance\_id**

instance ID. Either equal to 0, 1, 2, or 3.

**Type** int

**instrument\_abbr**

instrument abbreviation

**Type** str

**instrument\_full**

instrument encoded by its English name

**Type** str

**is\_resampled**

True if this sample was pitch-shifted from a neighbor; False if it was genuinely recorded.

**Type** bool

**pitch**

string containing English pitch class and octave number

**Type** str

**pitch\_id**

MIDI note index, where middle C (“C4”) corresponds to 60

**Type** int

**string\_id**

string ID. By musical convention, the first string is the highest. On wind instruments, this is replaced by *None*.

**Type** NoneType

**technique\_abbr**

playing technique abbreviation

**Type** str

**technique\_full**

playing technique encoded by its English name

**Type** str

**track\_id**

track id

**Type** str



**audio**

audio signal, sample rate

**Type** (np.ndarray, float)

**to\_jams()**

Jams: the track's data in jams format

**mirdata.tinysol.cite()**

Print the reference

**mirdata.tinysol.download**(*data\_home=None, partial\_download=None, force\_overwrite=False, cleanup=True*)

Download TinySOL.

**Parameters**

- **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*
- **force\_overwrite** (*bool*) – Whether to overwrite the existing downloaded data
- **partial\_download** (*list*) – List indicating what to partially download. The list can include any of: \* *'annotations'* the annotation files \* *'audio'* the audio files If *None*, all data is downloaded.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

**mirdata.tinysol.load**(*data\_home=None*)

Load TinySOL :param data\_home: Local path where TinySOL is stored.

If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns** {*track\_id*: track data}

**Return type** (dict)

**mirdata.tinysol.load\_audio**(*audio\_path*)

Load a TinySOL audio file.

**Parameters** **audio\_path** (*str*) – path to audio file

**Returns** the mono audio signal sr (float): The sample rate of the audio file

**Return type** y (np.ndarray)

**mirdata.tinysol.track\_ids**()

Return track ids

**Returns** A list of track ids

**Return type** (list)

**mirdata.tinysol.validate**(*data\_home=None, silence=False*)

Validate if the stored dataset is a valid version

**Parameters** **data\_home** (*str*) – Local path where the dataset is stored. If *None*, looks for the data in the default directory, *~/mir\_datasets*

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums** (list): **List of file paths that file exists in the dataset** index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (list)

## 2.3.2 Utilities

### `mirdata.track`

track object utility functions

**class** `mirdata.track.MultiTrack`

MultiTrack class.

A multitrack class is a collection of track objects and their associated audio that can be mixed together. A multitrack is itself a Track, and can have its own associated audio (such as a mastered mix), its own metadata and its own annotations.

**get\_mix()**

Create a linear mixture given a subset of tracks.

**Parameters** `track_keys` (list) – list of track keys to mix together

**Returns** mixture audio with shape (n\_samples, n\_channels)

**Return type** target (np.ndarray)

**get\_random\_target** (`n_tracks=None`, `min_weight=0.3`, `max_weight=1.0`)

Get a random target by combining a random selection of tracks with random weights

**Parameters**

- **n\_tracks** (*int* or *None*) – number of tracks to randomly mix. If *None*, uses all tracks
- **min\_weight** (*float*) – minimum possible weight when mixing
- **max\_weight** (*float*) – maximum possible weight when mixing

**Returns** mixture audio with shape (n\_samples, n\_channels) `tracks` (list): list of keys of included tracks `weights` (list): list of weights used to mix tracks

**Return type** target (np.ndarray)

**get\_target** (`track_keys`, `weights=None`, `average=True`, `enforce_length=True`)

Get target which is a linear mixture of tracks

**Parameters**

- **track\_keys** (list) – list of track keys to mix together
- **weights** (list or *None*) – list of positive scalars to be used in the average
- **average** (*bool*) – if *True*, computes a weighted average of the tracks if *False*, computes a weighted sum of the tracks
- **enforce\_length** (*bool*) – If *True*, raises `ValueError` if the tracks are not the same length. If *False*, pads audio with zeros to match the length of the longest track

**Returns** target audio with shape (n\_channels, n\_samples)

**Return type** target (np.ndarray)

**Raises** `ValueError` – if sample rates of the tracks are not equal if `enforce_length=True` and lengths are not equal

## mirdata.utils

Utility functions for mirdata

`mirdata.utils.MIR_DATASETS_DIR`  
home folder for MIR datasets

**Type** str

`mirdata.utils.NoteData`  
*intervals, notes, confidence*

**Type** namedtuple

`mirdata.utils.F0Data`  
*times, frequencies, confidence*

**Type** namedtuple

`mirdata.utils.LyricData`  
*start\_times, end\_times, lyrics, pronunciations*

**Type** namedtuple

`mirdata.utils.SectionData`  
*start\_times, end\_times, sections*

**Type** namedtuple

`mirdata.utils.BeatData`  
*beat\_times, beat\_positions*

**Type** namedtuple

`mirdata.utils.ChordData`  
*start\_times, end\_times, chords*

**Type** namedtuple

`mirdata.utils.KeyData`  
*start\_times, end\_times, keys*

**Type** namedtuple

`mirdata.utils.EventData`  
*start\_times, end\_times, event*

**Type** namedtuple

`mirdata.utils.TempoData`  
*time, duration, value, confidence*

**Type** namedtuple

**class** `mirdata.utils.BeatData` (*beat\_times, beat\_positions*)

**beat\_positions**  
Alias for field number 1

**beat\_times**  
Alias for field number 0

**class** `mirdata.utils.ChordData` (*intervals, labels*)

**intervals**

Alias for field number 0

**labels**

Alias for field number 1

**class** mirdata.utils.**EventData** (*start\_times, end\_times, event*)

**end\_times**

Alias for field number 1

**event**

Alias for field number 2

**start\_times**

Alias for field number 0

**class** mirdata.utils.**F0Data** (*times, frequencies, confidence*)

**confidence**

Alias for field number 2

**frequencies**

Alias for field number 1

**times**

Alias for field number 0

**class** mirdata.utils.**KeyData** (*start\_times, end\_times, keys*)

**end\_times**

Alias for field number 1

**keys**

Alias for field number 2

**start\_times**

Alias for field number 0

**class** mirdata.utils.**LyricData** (*start\_times, end\_times, lyrics, pronunciations*)

**end\_times**

Alias for field number 1

**lyrics**

Alias for field number 2

**pronunciations**

Alias for field number 3

**start\_times**

Alias for field number 0

**class** mirdata.utils.**MultipitchData** (*times, frequency\_list, confidence\_list*)

**confidence\_list**

Alias for field number 2

**frequency\_list**

Alias for field number 1

**times**

Alias for field number 0

**class** `mirdata.utils.NoteData` (*intervals, notes, confidence*)

**confidence**

Alias for field number 2

**intervals**

Alias for field number 0

**notes**

Alias for field number 1

**class** `mirdata.utils.SectionData` (*intervals, labels*)

**intervals**

Alias for field number 0

**labels**

Alias for field number 1

**class** `mirdata.utils.TempoData` (*time, duration, value, confidence*)

**confidence**

Alias for field number 3

**duration**

Alias for field number 1

**time**

Alias for field number 0

**value**

Alias for field number 2

**class** `mirdata.utils.cached_property` (*func*)

A property that is only computed once per instance and then replaces itself with an ordinary attribute. Deleting the attribute resets the property. Source: <https://github.com/bottlepy/bottle/commit/fa7733e075da0d790d809aa3d2f53071897e6f76>

`mirdata.utils.check_index` (*dataset\_index, data\_home*)

check index to find out missing files and files with invalid checksum

**Parameters**

- **dataset\_index** (*list*) – dataset indices
- **data\_home** (*str*) – Local home path that the dataset is being stored

**Returns**

**List of file paths that are in the dataset index** but missing locally

**invalid\_checksums (list):** List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum

**Return type** `missing_files` (*list*)

`mirdata.utils.get_default_dataset_path(dataset_name)`

Get the default path for a dataset given it's name

**Parameters** `dataset_name` (*str* or *None*) – The name of the dataset folder, e.g. 'Orchset'

**Returns** Local path to the dataset

**Return type** `save_path` (*str*)

`mirdata.utils.log_message(message, silence=False)`

Helper function to log message

**Parameters**

- **message** (*str*) – message to log
- **silence** (*bool*) – if true, the message is not logged

`mirdata.utils.md5(file_path)`

Get md5 hash of a file.

**Parameters** `file_path` (*str*) – File path

**Returns** md5 hash of data in `file_path`

**Return type** `md5_hash` (*str*)

`mirdata.utils.none_path_join(partial_path_list)`

Join a list of partial paths. If any part of the path is None, returns None.

**Parameters** `partial_path_list` (*list*) – List of partial paths

**Returns** joined path string or None

**Return type** `path` or `None` (*str* or *None*)

`mirdata.utils.validator(dataset_index, data_home, silence=False)`

Checks the existence and validity of files stored locally with respect to the paths and file checksums stored in the reference index. Logs invalid checksums and missing files.

**Parameters**

- **dataset\_index** (*list*) – dataset indices
- **data\_home** (*str*) – Local home path that the dataset is being stored
- **silence** (*bool*) – if False (default), prints missing and invalid files
- **stdout**. Otherwise, this function is equivalent to `check_index.to` –

**Returns**

**List of file paths that are in the dataset index** but missing locally.

**invalid\_checksums (list):** List of file paths that file exists in the dataset index but has a different checksum compare to the reference checksum.

**Return type** `missing_files` (*list*)

## mirdata.download\_utils

functions for downloading from the web

`mirdata.download_utils.RemoteFileMetadata`

It specifies the metadata of the remote file to download. The metadata consists of *filename*, *url*, *checksum*, and *destination\_dir*.

**Type** namedtuple

```
class mirdata.download_utils.DownloadProgressBar(iterable=None, desc=None, total=None, leave=True, file=None, ncols=None, mininterval=0.1, maxinterval=10.0, miniters=None, ascii=None, disable=False, unit='it', unit_scale=False, dynamic_ncols=False, smoothing=0.3, bar_format=None, initial=0, position=None, postfix=None, unit_divisor=1000, write_bytes=None, lock_args=None, nrows=None, colour=None, gui=False, **kwargs)
```

Wrap *tqdm* to show download progress

```
class mirdata.download_utils.RemoteFileMetadata(filename, url, checksum, destination_dir)
```

**checksum**

Alias for field number 2

**destination\_dir**

Alias for field number 3

**filename**

Alias for field number 0

**url**

Alias for field number 1

`mirdata.download_utils.download_from_remote(remote, save_dir, force_overwrite=False)`

Download a remote dataset into path Fetch a dataset pointed by remote's url, save into path using remote's filename and ensure its integrity based on the MD5 Checksum of the downloaded file.

Adapted from scikit-learn's `sklearn.datasets.base._fetch_remote`.

**Parameters**

- **remote** (*RemoteFileMetadata*) – Named tuple containing remote dataset meta information: url, filename and checksum
- **save\_dir** (*str*) – Directory to save the file to. Usually *data\_home*
- **force\_overwrite** (*bool*) – If True, overwrite existing file with the downloaded file. If False, does not overwrite, but checks that checksum is consistent.

**Returns** Full path of the created file.

**Return type** *file\_path* (*str*)

```
mirdata.download_utils.download_tar_file(tar_remote, save_dir, force_overwrite, cleanup=True)
```

Download and untar a tar file.

**Parameters**

- **tar\_remote** (*RemoteFileMetadata*) – Object containing download information

- **save\_dir** (*str*) – Path to save downloaded file
- **force\_overwrite** (*bool*) – If True, overwrites existing files
- **cleanup** (*bool*) – If True, remove tarfile after untarring. Default=False

`mirdata.download_utils.download_zip_file(zip_remote, save_dir, force_overwrite, cleanup=True)`

Download and unzip a zip file.

#### Parameters

- **zip\_remote** (*RemoteFileMetadata*) – Object containing download information
- **save\_dir** (*str*) – Path to save downloaded file
- **force\_overwrite** (*bool*) – If True, overwrites existing files
- **cleanup** (*bool*) – If True, remove zipfile after unzipping. Default=False

`mirdata.download_utils.downloader(save_dir, remotes=None, partial_download=None, info_message=None, force_overwrite=False, cleanup=True)`

Download data to *save\_dir* and optionally print a message.

#### Parameters

- **save\_dir** (*str*) – The directory to download the data
- **remotes** (*dict or None*) – A dictionary of *RemoteFileMetadata* tuples of data in zip format. If None, there is no data to download
- **partial\_download** (*list or None*) – A list of keys to partially download the remote objects of the download dict. If None, all data is downloaded
- **info\_message** (*str or None*) – A string of info to print when this function is called. If None, no string is printed.
- **force\_overwrite** (*bool*) – If True, existing files are overwritten by the downloaded files.
- **cleanup** (*bool*) – Whether to delete the zip/tar file after extracting.

`mirdata.download_utils.untar(tar_path, cleanup=True)`

Untar a tar file inside it's current directory.

#### Parameters

- **tar\_path** (*str*) – Path to tar file
- **cleanup** (*bool*) – If True, remove tarfile after untarring. Default=False

`mirdata.download_utils.unzip(zip_path, cleanup=True)`

Unzip a zip file inside it's current directory.

#### Parameters

- **zip\_path** (*str*) – Path to zip file
- **cleanup** (*bool*) – If True, remove zipfile after unzipping. Default=False

## mirdata.jams\_utils

functions for converting mirdata annotations to jams format



`mirdata.jams_utils.beats_to_jams` (*beats*)

Convert beats annotations into jams format.

**Parameters** `beats` (*tuple*) – A tuple in the format (BeatData, str), where str describes the annotation and BeatData is the beats mirdata annotation format.

**Returns** `jannot_beat`

**Return type** JAM beat annotation object.

`mirdata.jams_utils.chords_to_jams` (*chords*)

Convert chords annotations into jams format.

**Parameters** `chords` (*tuple*) – A tuple in the format (ChordData, str), where str describes the annotation and ChordData is the chords mirdata annotation format.

**Returns** `jannot_chord`

**Return type** JAM chord annotation object.

`mirdata.jams_utils.events_to_jams` (*events*)

Convert events annotations into jams format.

**Parameters** `events` (*tuple*) – A tuple in the format (EventData, str), where str describes the annotation and EventData is the events mirdata annotation format.

**Returns** `jannot_events`

**Return type** JAM tag\_open annotation object.

`mirdata.jams_utils.f0s_to_jams` (*f0s*)

Convert f0 annotations into jams format.

**Parameters** `f0s` (*tuple*) – A tuple in the format (F0Data, str), where str describes the annotation and F0Data is the f0 mirdata annotation format.

**Returns** `jannot_f0`

**Return type** JAM pitch\_contour annotation object.

`mirdata.jams_utils.jams_converter` (*audio\_path=None, beat\_data=None, chord\_data=None, note\_data=None, f0\_data=None, section\_data=None, multi\_section\_data=None, tempo\_data=None, event\_data=None, key\_data=None, lyrics\_data=None, tags\_gtzan\_data=None, metadata=None*)

Convert annotations from a track to JAMS format.

**Parameters**

- **(str or None)** (*audio\_path*) – A path to the corresponding audio file, or None. If provided, the audio file will be read to compute the duration. If None, ‘duration’ must be a field in the metadata dictionary, or the resulting jam object will not validate.
- **(list or None)** (*tags\_gtzan\_data*) – A list of tuples of (BeatData, str), where str describes the annotation (e.g. ‘beats\_1’).
- **(list or None)** – A list of tuples of (ChordData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (NoteData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (F0Data, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (SectionData, str), where str describes the annotation.

- **(list or None)** – A list of tuples. Tuples in `multi_section_data` should contain another list of tuples, indicating annotations in the different levels e.g. `((segments0, level0), (segments1, level1), annotator)` and a str indicating the annotator
- **(list or None)** – A list of tuples of (float, str), where float gives the tempo in bpm and str describes the annotation.
- **(list or None)** – A list of tuples of (EventData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (KeyData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (LyricData, str), where str describes the annotation.
- **(list or None)** – A list of tuples of (str, str), where the first str is the tag and the second is a descriptor of the annotation.
- **(dict or None)** (*metadata*) – A dictionary containing the track metadata.

**Returns** `jam` – A JAM object with all the annotations.

**Return type** JAM object

`mirdata.jams_utils.keys_to_jams(keys)`

Convert keys annotations into jams format.

**Parameters** `keys` (*tuple*) – A tuple in the format (KeyData, str), where str describes the annotation and KeyData is the keys mirdata annotation format.

**Returns** `jannot_key`

**Return type** JAM key\_mode annotation object.

`mirdata.jams_utils.lyrics_to_jams(lyrics)`

Convert lyrics annotations into jams format.

**Parameters** `lyrics` (*tuple*) – A tuple in the format (LyricData, str), where str describes the annotation and LyricData is the lyric mirdata annotation format.

**Returns** `jannot_lyric`

**Return type** JAM lyric annotation object.

`mirdata.jams_utils.multi_sections_to_jams(multi_sections)`

Convert hierarchical annotations into jams format.

**Parameters** `multi_segment` (*list*) – A list of tuples in the format `((segments0, level0), (segments1, level1), annotator)`, where segments are SectionData mirdata format, level indicates the hierarchy (e.g. 0, 1) and annotator describes the annotator. This format is customize for Salami dataset annotations.

**Returns** `jannot_multi`

**Return type** JAM multi\_segment annotation object.

`mirdata.jams_utils.notes_to_jams(notes)`

Convert notes annotations into jams format using `note_to_midi` from `librosa`.

**Parameters** `notes` (*tuple*) – A tuple in the format (NoteData, str), where str describes the annotation and NoteData is the notes mirdata annotation format.

**Returns** `jannot_notes`

**Return type** JAM note\_midi annotation object.

`mirdata.jams_utils.sections_to_jams` (*sections*)

Convert sections annotations into jams format.

**Parameters** `sections` (*tuple*) – A tuple in the format (SectionData, str), where str describes the annotation and SectionData is the sections mirdata annotation format.

**Returns** `jannot_seg`

**Return type** JAM segment\_open annotation object.

`mirdata.jams_utils.tag_gtzan_to_jams` (*tags*)

Convert tag-gtzan annotations into jams format.

**Parameters** `tags` (*tuple*) – A tuple in the format (str, str), where the first str is the tag and the second describes the annotation.

**Returns** `jannot_tag_gtzan`

**Return type** JAM tag\_gtzan annotation object.

`mirdata.jams_utils.tempos_to_jams` (*tempos*)

Convert tempo annotations into jams format.

**Parameters** `tempo` (*tuple*) – A tuple in the format (float, str), where str describes the annotation and float is the tempo in beats per minute.

**Returns** `jannot_tempo`

**Return type** JAM tempo annotation object.

## 2.4 FAQ

### 2.4.1 How do I add a new loader?

Take a look at our [instructions](#)!

### 2.4.2 How do I get access to a dataset if the download function says it's not available?

We don't distribute data ourselves, so unfortunately it is up to you to find the data yourself. We strongly encourage you to favor datasets which are currently available.

### 2.4.3 Can you send me the data for a dataset which is not available?

No, we do not host or distribute datasets.

### 2.4.4 How do I request a new dataset?

Open an [issue](#) and tag it with the “New Loader” label.

### **2.4.5 What do I do if my data fails validation?**

Very often, data fails validation because of how the files are named or how the folder is structured. If this is the case, try renaming/reorganizing your data to match what mirdata expects. If your data fails validation because of the checksums, this means that you are using data which is different from what most people are using, and you should try to get the more common dataset version, for example by using the data loader's download function.

### **2.4.6 How do you choose the data that is used to create the checksums?**

Whenever possible, the data downloaded using `.download()` is the same data used to create the checksums. If this isn't possible, we did our best to get the data from the original source (the dataset creator) in order to create the checksum. If this is again not possible, we found as many versions of the data as we could from different users of the dataset, computed checksums on all of them and used the version which was the most common amongst them.

### **2.4.7 Does mirdata provide data loaders for pytorch/Tensorflow?**

For now, no. Music datasets are very widely varied in their annotation types and supported tasks. To make a data loader, there would need to be "standard" ways to encode the desired inputs/outputs - unfortunately this is not universal for most datasets and usages. Still, this library provides the necessary first step for building data loaders and it is easy to build data loaders on top of this. For a simple example, see our [examples](#) page.

### **2.4.8 Why didn't you release a version of this library in MATLAB/C/Java/R?**

The creators of this library are Python users, so we made a library in python. We'd be very happy to provide guidance to anyone who wants to create a version of this library in another programming languages.

### **2.4.9 A download link is broken for a loader's `.download()` function. What do I do?**

Please open an [issue](#) and tag it with the "broken link" label.

### **2.4.10 Why the name, mirdata?**

mirdata = mir + data. MIR is an acronym for Music Information Retrieval, and the library was built for working with data.

### **2.4.11 If I find a mistake in an annotation, should I fix it in the loader?**

No. All datasets have "mistakes", and we do not want to create another version of each dataset ourselves. The loaders should load the data as released. After that, it's up to the user what they want to do with it.

### **2.4.12 Does mirdata support data which lives off-disk?**

Yes. While the simple usage of mirdata assumes that data lives on-disk, it can be used for off-disk data as well. See the "local vs remote" example in the [examples](#) page for details.

## CHAPTER 3

---

### Contribute

---

- [Issue Tracker](#)
- [Source Code](#)



### m

- `mirdata`, 12
- `mirdata.beatles`, 12
- `mirdata.beatport_key`, 14
- `mirdata.dali`, 17
- `mirdata.download_utils`, 66
- `mirdata.giantsteps_key`, 23
- `mirdata.giantsteps_tempo`, 20
- `mirdata.groove_midi`, 26
- `mirdata.gtzan_genre`, 29
- `mirdata.guitarset`, 30
- `mirdata.ikala`, 35
- `mirdata.jams_utils`, 68
- `mirdata.maestro`, 37
- `mirdata.medley_solos_db`, 44
- `mirdata.medleydb_melody`, 40
- `mirdata.medleydb_pitch`, 42
- `mirdata.orchset`, 46
- `mirdata.rwc_classical`, 49
- `mirdata.rwc_jazz`, 51
- `mirdata.rwc_popular`, 54
- `mirdata.salami`, 56
- `mirdata.tinysol`, 59
- `mirdata.track`, 62
- `mirdata.utils`, 63





## A

- album (*mirdata.dali.Track attribute*), 17
- alternating\_melody (*mirdata.orchset.Track attribute*), 46
- annotation\_object (*mirdata.dali.Track attribute*), 18
- annotation\_path (*mirdata.dali.Track attribute*), 17
- annotation\_v1\_path (*mirdata.giantsteps\_tempo.Track attribute*), 21
- annotation\_v2\_path (*mirdata.giantsteps\_tempo.Track attribute*), 21
- annotator\_1\_id (*mirdata.salami.Track attribute*), 57
- annotator\_1\_time (*mirdata.salami.Track attribute*), 57
- annotator\_2\_id (*mirdata.salami.Track attribute*), 57
- annotator\_2\_time (*mirdata.salami.Track attribute*), 57
- artist (*mirdata.dali.Track attribute*), 18
- artist (*mirdata.medleydb\_melody.Track attribute*), 40
- artist (*mirdata.medleydb\_pitch.Track attribute*), 42
- artist (*mirdata.rwc\_classical.Track attribute*), 49
- artist (*mirdata.rwc\_jazz.Track attribute*), 52
- artist (*mirdata.rwc\_popular.Track attribute*), 54
- artist (*mirdata.salami.Track attribute*), 57
- artists (*mirdata.beatport\_key.Track attribute*), 15
- artists (*mirdata.giantsteps\_key.Track attribute*), 24
- audio (*mirdata.beatles.Track attribute*), 13
- audio (*mirdata.beatport\_key.Track attribute*), 15
- audio (*mirdata.dali.Track attribute*), 19
- audio (*mirdata.giantsteps\_key.Track attribute*), 24
- audio (*mirdata.giantsteps\_tempo.Track attribute*), 21
- audio (*mirdata.groove\_midi.Track attribute*), 27
- audio (*mirdata.gtzan\_genre.Track attribute*), 29
- audio (*mirdata.maestro.Track attribute*), 38
- audio (*mirdata.medley\_solos\_db.Track attribute*), 45
- audio (*mirdata.medleydb\_melody.Track attribute*), 41
- audio (*mirdata.medleydb\_pitch.Track attribute*), 43
- audio (*mirdata.rwc\_classical.Track attribute*), 50
- audio (*mirdata.rwc\_jazz.Track attribute*), 52
- audio (*mirdata.rwc\_popular.Track attribute*), 55
- audio (*mirdata.salami.Track attribute*), 58
- audio (*mirdata.tinysol.Track attribute*), 60
- audio\_hex (*mirdata.guitarset.Track attribute*), 32
- audio\_hex\_cln (*mirdata.guitarset.Track attribute*), 32
- audio\_hex\_cln\_path (*mirdata.guitarset.Track attribute*), 31
- audio\_hex\_path (*mirdata.guitarset.Track attribute*), 31
- audio\_mic (*mirdata.guitarset.Track attribute*), 32
- audio\_mic\_path (*mirdata.guitarset.Track attribute*), 31
- audio\_mix (*mirdata.guitarset.Track attribute*), 32
- audio\_mix\_path (*mirdata.guitarset.Track attribute*), 31
- audio\_mono (*mirdata.orchset.Track attribute*), 47
- audio\_path (*mirdata.beatles.Track attribute*), 12
- audio\_path (*mirdata.beatport\_key.Track attribute*), 15
- audio\_path (*mirdata.dali.Track attribute*), 18
- audio\_path (*mirdata.giantsteps\_key.Track attribute*), 23
- audio\_path (*mirdata.giantsteps\_tempo.Track attribute*), 21
- audio\_path (*mirdata.groove\_midi.Track attribute*), 27
- audio\_path (*mirdata.gtzan\_genre.Track attribute*), 29
- audio\_path (*mirdata.ikala.Track attribute*), 35
- audio\_path (*mirdata.maestro.Track attribute*), 37
- audio\_path (*mirdata.medley\_solos\_db.Track attribute*), 44
- audio\_path (*mirdata.medleydb\_melody.Track attribute*), 40
- audio\_path (*mirdata.medleydb\_pitch.Track attribute*), 42
- audio\_path (*mirdata.rwc\_classical.Track attribute*), 49
- audio\_path (*mirdata.rwc\_jazz.Track attribute*), 52
- audio\_path (*mirdata.rwc\_popular.Track attribute*), 54
- audio\_path (*mirdata.salami.Track attribute*), 57
- audio\_path (*mirdata.tinysol.Track attribute*), 59

audio\_path\_mono (*mirdata.orchset.Track* attribute), 46  
 audio\_path\_stereo (*mirdata.orchset.Track* attribute), 46  
 audio\_stereo (*mirdata.orchset.Track* attribute), 47  
 audio\_url (*mirdata.dali.Track* attribute), 18

## B

beat\_positions (*mirdata.utils.BeatData* attribute), 63  
 beat\_times (*mirdata.utils.BeatData* attribute), 63  
 beat\_type (*mirdata.groove\_midi.Track* attribute), 27  
 BeatData (class in *mirdata.utils*), 63  
 BeatData (in module *mirdata.utils*), 63  
 beats (*mirdata.beatles.Track* attribute), 13  
 beats (*mirdata.groove\_midi.Track* attribute), 27  
 beats (*mirdata.guitarset.Track* attribute), 32  
 beats (*mirdata.rwc\_classical.Track* attribute), 50  
 beats (*mirdata.rwc\_jazz.Track* attribute), 52  
 beats (*mirdata.rwc\_popular.Track* attribute), 55  
 beats\_path (*mirdata.beatles.Track* attribute), 12  
 beats\_path (*mirdata.rwc\_classical.Track* attribute), 49  
 beats\_path (*mirdata.rwc\_jazz.Track* attribute), 52  
 beats\_path (*mirdata.rwc\_popular.Track* attribute), 54  
 beats\_to\_jams() (in module *mirdata.jams\_utils*), 68  
 broad\_genre (*mirdata.salami.Track* attribute), 57

## C

cached\_property (class in *mirdata.utils*), 65  
 canonical\_composer (*mirdata.maestro.Track* attribute), 37  
 canonical\_title (*mirdata.maestro.Track* attribute), 37  
 category (*mirdata.rwc\_classical.Track* attribute), 49  
 check\_index() (in module *mirdata.utils*), 65  
 checksum (*mirdata.download\_utils.RemoteFileMetadata* attribute), 67  
 ChordData (class in *mirdata.utils*), 63  
 ChordData (in module *mirdata.utils*), 63  
 chords (*mirdata.beatles.Track* attribute), 13  
 chords (*mirdata.rwc\_popular.Track* attribute), 55  
 chords\_path (*mirdata.beatles.Track* attribute), 12  
 chords\_path (*mirdata.rwc\_popular.Track* attribute), 54  
 chords\_to\_jams() (in module *mirdata.jams\_utils*), 69  
 cite() (in module *mirdata.beatles*), 13  
 cite() (in module *mirdata.beatport\_key*), 16  
 cite() (in module *mirdata.dali*), 19  
 cite() (in module *mirdata.giantsteps\_key*), 24  
 cite() (in module *mirdata.giantsteps\_tempo*), 22  
 cite() (in module *mirdata.groove\_midi*), 27  
 cite() (in module *mirdata.gtzan\_genre*), 29

cite() (in module *mirdata.guitarset*), 33  
 cite() (in module *mirdata.ikala*), 36  
 cite() (in module *mirdata.maestro*), 38  
 cite() (in module *mirdata.medley\_solos\_db*), 45  
 cite() (in module *mirdata.medleydb\_melody*), 41  
 cite() (in module *mirdata.medleydb\_pitch*), 43  
 cite() (in module *mirdata.orchset*), 47  
 cite() (in module *mirdata.salami*), 58  
 cite() (in module *mirdata.tinysol*), 61  
 composer (*mirdata.orchset.Track* attribute), 46  
 composer (*mirdata.rwc\_classical.Track* attribute), 49  
 confidence (*mirdata.utils.F0Data* attribute), 64  
 confidence (*mirdata.utils.NoteData* attribute), 65  
 confidence (*mirdata.utils.TempoData* attribute), 65  
 confidence\_list (*mirdata.utils.MultipitchData* attribute), 64  
 contains\_brass (*mirdata.orchset.Track* attribute), 46  
 contains\_strings (*mirdata.orchset.Track* attribute), 46  
 contains\_winds (*mirdata.orchset.Track* attribute), 46

## D

dataset\_version (*mirdata.dali.Track* attribute), 18  
 destination\_dir (*mirdata.download\_utils.RemoteFileMetadata* attribute), 67  
 download() (in module *mirdata.beatles*), 13  
 download() (in module *mirdata.beatport\_key*), 16  
 download() (in module *mirdata.dali*), 19  
 download() (in module *mirdata.giantsteps\_key*), 24  
 download() (in module *mirdata.giantsteps\_tempo*), 22  
 download() (in module *mirdata.groove\_midi*), 27  
 download() (in module *mirdata.gtzan\_genre*), 29  
 download() (in module *mirdata.guitarset*), 33  
 download() (in module *mirdata.ikala*), 36  
 download() (in module *mirdata.maestro*), 38  
 download() (in module *mirdata.medley\_solos\_db*), 45  
 download() (in module *mirdata.medleydb\_melody*), 41  
 download() (in module *mirdata.medleydb\_pitch*), 43  
 download() (in module *mirdata.orchset*), 47  
 download() (in module *mirdata.rwc\_classical*), 50  
 download() (in module *mirdata.rwc\_jazz*), 53  
 download() (in module *mirdata.rwc\_popular*), 55  
 download() (in module *mirdata.salami*), 58  
 download() (in module *mirdata.tinysol*), 61  
 download\_from\_remote() (in module *mirdata.download\_utils*), 67  
 download\_tar\_file() (in module *mirdata.download\_utils*), 67  
 download\_zip\_file() (in module *mirdata.download\_utils*), 68

[downloader\(\)](#) (in module *mirdata.download\_utils*), 68  
[DownloadProgressBar](#) (class in *mirdata.download\_utils*), 67  
[drum\\_events](#) (*mirdata.groove\_midi.Track* attribute), 27  
[drum\\_information](#) (*mirdata.rwc\_popular.Track* attribute), 54  
[drummer](#) (*mirdata.groove\_midi.Track* attribute), 26  
[duration](#) (*mirdata.groove\_midi.Track* attribute), 27  
[duration](#) (*mirdata.maestro.Track* attribute), 38  
[duration](#) (*mirdata.rwc\_classical.Track* attribute), 49  
[duration](#) (*mirdata.rwc\_jazz.Track* attribute), 52  
[duration](#) (*mirdata.rwc\_popular.Track* attribute), 54  
[duration](#) (*mirdata.salami.Track* attribute), 57  
[duration](#) (*mirdata.utils.TempoData* attribute), 65  
[dynamics](#) (*mirdata.tinysol.Track* attribute), 59  
[dynamics\\_id](#) (*mirdata.tinysol.Track* attribute), 60

## E

[end\\_times](#) (*mirdata.utils.EventData* attribute), 64  
[end\\_times](#) (*mirdata.utils.KeyData* attribute), 64  
[end\\_times](#) (*mirdata.utils.LyricData* attribute), 64  
[event](#) (*mirdata.utils.EventData* attribute), 64  
[EventData](#) (class in *mirdata.utils*), 64  
[EventData](#) (in module *mirdata.utils*), 63  
[events\\_to\\_jams\(\)](#) (in module *mirdata.jams\_utils*), 69  
[excerpt](#) (*mirdata.orchset.Track* attribute), 47

## F

[f0](#) (*mirdata.ikala.Track* attribute), 35  
[f0\\_path](#) (*mirdata.ikala.Track* attribute), 35  
[F0Data](#) (class in *mirdata.utils*), 64  
[F0Data](#) (in module *mirdata.utils*), 63  
[f0s\\_to\\_jams\(\)](#) (in module *mirdata.jams\_utils*), 69  
[family](#) (*mirdata.tinysol.Track* attribute), 60  
[filename](#) (*mirdata.download\_utils.RemoteFileMetadata* attribute), 67  
[find\\_replace\(\)](#) (in module *mirdata.beatport\_key*), 16  
[frequencies](#) (*mirdata.utils.F0Data* attribute), 64  
[frequency\\_list](#) (*mirdata.utils.MultipitchData* attribute), 64

## G

[genre](#) (*mirdata.giantsteps\_tempo.Track* attribute), 21  
[genre](#) (*mirdata.gtzan\_genre.Track* attribute), 29  
[genre](#) (*mirdata.medleydb\_melody.Track* attribute), 40  
[genre](#) (*mirdata.medleydb\_pitch.Track* attribute), 42  
[genre](#) (*mirdata.salami.Track* attribute), 57  
[genres](#) (*mirdata.beatport\_key.Track* attribute), 15  
[genres](#) (*mirdata.giantsteps\_key.Track* attribute), 24

[get\\_default\\_dataset\\_path\(\)](#) (in module *mirdata.utils*), 65  
[get\\_mix\(\)](#) (*mirdata.track.MultiTrack* method), 62  
[get\\_random\\_target\(\)](#) (*mirdata.track.MultiTrack* method), 62  
[get\\_target\(\)](#) (*mirdata.track.MultiTrack* method), 62  
[ground\\_truth](#) (*mirdata.dali.Track* attribute), 18

## I

[inferred\\_chords](#) (*mirdata.guitarset.Track* attribute), 32  
[instance\\_id](#) (*mirdata.tinysol.Track* attribute), 60  
[instrument](#) (*mirdata.medley\_solos\_db.Track* attribute), 44  
[instrument](#) (*mirdata.medleydb\_pitch.Track* attribute), 42  
[instrument\\_abbr](#) (*mirdata.tinysol.Track* attribute), 60  
[instrument\\_full](#) (*mirdata.tinysol.Track* attribute), 60  
[instrument\\_id](#) (*mirdata.medley\_solos\_db.Track* attribute), 44  
[instrumental\\_audio](#) (*mirdata.ikala.Track* attribute), 35  
[instruments](#) (*mirdata.rwc\_jazz.Track* attribute), 52  
[instruments](#) (*mirdata.rwc\_popular.Track* attribute), 54  
[intervals](#) (*mirdata.utils.ChordData* attribute), 63  
[intervals](#) (*mirdata.utils.NoteData* attribute), 65  
[intervals](#) (*mirdata.utils.SectionData* attribute), 65  
[is\\_excerpt](#) (*mirdata.medleydb\_melody.Track* attribute), 40  
[is\\_instrumental](#) (*mirdata.medleydb\_melody.Track* attribute), 40  
[is\\_resampled](#) (*mirdata.tinysol.Track* attribute), 60

## J

[jams\\_converter\(\)](#) (in module *mirdata.jams\_utils*), 69  
[jams\\_path](#) (*mirdata.guitarset.Track* attribute), 31

## K

[key](#) (*mirdata.beatles.Track* attribute), 13  
[key](#) (*mirdata.beatport\_key.Track* attribute), 15  
[key](#) (*mirdata.giantsteps\_key.Track* attribute), 24  
[key\\_mode](#) (*mirdata.guitarset.Track* attribute), 32  
[KeyData](#) (class in *mirdata.utils*), 64  
[KeyData](#) (in module *mirdata.utils*), 63  
[keys](#) (*mirdata.utils.KeyData* attribute), 64  
[keys\\_path](#) (*mirdata.beatles.Track* attribute), 12  
[keys\\_path](#) (*mirdata.beatport\_key.Track* attribute), 15  
[keys\\_path](#) (*mirdata.giantsteps\_key.Track* attribute), 23  
[keys\\_to\\_jams\(\)](#) (in module *mirdata.jams\_utils*), 70

## L

[labels \(mirdata.utils.ChordData attribute\), 64](#)  
[labels \(mirdata.utils.SectionData attribute\), 65](#)  
[language \(mirdata.dali.Track attribute\), 18](#)  
[leadsheet\\_chords \(mirdata.guitarset.Track attribute\), 32](#)  
[lines \(mirdata.dali.Track attribute\), 19](#)  
[load \(\) \(in module mirdata.beatles\), 13](#)  
[load \(\) \(in module mirdata.beatport\\_key\), 16](#)  
[load \(\) \(in module mirdata.dali\), 19](#)  
[load \(\) \(in module mirdata.giantsteps\\_key\), 24](#)  
[load \(\) \(in module mirdata.giantsteps\\_tempo\), 22](#)  
[load \(\) \(in module mirdata.groove\\_midi\), 28](#)  
[load \(\) \(in module mirdata.gtzan\\_genre\), 30](#)  
[load \(\) \(in module mirdata.guitarset\), 33](#)  
[load \(\) \(in module mirdata.ikala\), 36](#)  
[load \(\) \(in module mirdata.maestro\), 39](#)  
[load \(\) \(in module mirdata.medley\\_solos\\_db\), 45](#)  
[load \(\) \(in module mirdata.medleydb\\_melody\), 41](#)  
[load \(\) \(in module mirdata.medleydb\\_pitch\), 43](#)  
[load \(\) \(in module mirdata.orchset\), 48](#)  
[load \(\) \(in module mirdata.rwc\\_classical\), 50](#)  
[load \(\) \(in module mirdata.rwc\\_jazz\), 53](#)  
[load \(\) \(in module mirdata.rwc\\_popular\), 56](#)  
[load \(\) \(in module mirdata.salami\), 58](#)  
[load \(\) \(in module mirdata.tinysol\), 61](#)  
[load\\_annotations\\_class \(\) \(in module mirdata.dali\), 19](#)  
[load\\_annotations\\_granularity \(\) \(in module mirdata.dali\), 19](#)  
[load\\_artist \(\) \(in module mirdata.beatport\\_key\), 16](#)  
[load\\_artist \(\) \(in module mirdata.giantsteps\\_key\), 25](#)  
[load\\_audio \(\) \(in module mirdata.beatles\), 13](#)  
[load\\_audio \(\) \(in module mirdata.beatport\\_key\), 16](#)  
[load\\_audio \(\) \(in module mirdata.dali\), 19](#)  
[load\\_audio \(\) \(in module mirdata.giantsteps\\_key\), 25](#)  
[load\\_audio \(\) \(in module mirdata.giantsteps\\_tempo\), 22](#)  
[load\\_audio \(\) \(in module mirdata.groove\\_midi\), 28](#)  
[load\\_audio \(\) \(in module mirdata.gtzan\\_genre\), 30](#)  
[load\\_audio \(\) \(in module mirdata.guitarset\), 33](#)  
[load\\_audio \(\) \(in module mirdata.maestro\), 39](#)  
[load\\_audio \(\) \(in module mirdata.medley\\_solos\\_db\), 45](#)  
[load\\_audio \(\) \(in module mirdata.medleydb\\_melody\), 41](#)  
[load\\_audio \(\) \(in module mirdata.medleydb\\_pitch\), 43](#)  
[load\\_audio \(\) \(in module mirdata.rwc\\_classical\), 51](#)  
[load\\_audio \(\) \(in module mirdata.salami\), 58](#)  
[load\\_audio \(\) \(in module mirdata.tinysol\), 61](#)  
[load\\_audio\\_mono \(\) \(in module mirdata.orchset\), 48](#)

[load\\_audio\\_stereo \(\) \(in module mirdata.orchset\), 48](#)  
[load\\_beats \(\) \(in module mirdata.beatles\), 13](#)  
[load\\_beats \(\) \(in module mirdata.groove\\_midi\), 28](#)  
[load\\_chords \(\) \(in module mirdata.beatles\), 14](#)  
[load\\_chords \(\) \(in module mirdata.guitarset\), 34](#)  
[load\\_drum\\_events \(\) \(in module mirdata.groove\\_midi\), 28](#)  
[load\\_genre \(\) \(in module mirdata.beatport\\_key\), 16](#)  
[load\\_genre \(\) \(in module mirdata.giantsteps\\_key\), 25](#)  
[load\\_genre \(\) \(in module mirdata.giantsteps\\_tempo\), 22](#)  
[load\\_instrumental\\_audio \(\) \(in module mirdata.ikala\), 36](#)  
[load\\_key \(\) \(in module mirdata.beatles\), 14](#)  
[load\\_key \(\) \(in module mirdata.beatport\\_key\), 17](#)  
[load\\_key \(\) \(in module mirdata.giantsteps\\_key\), 25](#)  
[load\\_midi \(\) \(in module mirdata.groove\\_midi\), 28](#)  
[load\\_midi \(\) \(in module mirdata.maestro\), 39](#)  
[load\\_mix\\_audio \(\) \(in module mirdata.ikala\), 36](#)  
[load\\_multitrack\\_audio \(\) \(in module mirdata.guitarset\), 34](#)  
[load\\_note\\_ann \(\) \(in module mirdata.guitarset\), 34](#)  
[load\\_notes \(\) \(in module mirdata.maestro\), 39](#)  
[load\\_pitch\\_contour \(\) \(in module mirdata.guitarset\), 34](#)  
[load\\_sections \(\) \(in module mirdata.beatles\), 14](#)  
[load\\_tempo \(\) \(in module mirdata.beatport\\_key\), 17](#)  
[load\\_tempo \(\) \(in module mirdata.giantsteps\\_key\), 25](#)  
[load\\_tempo \(\) \(in module mirdata.giantsteps\\_tempo\), 22](#)  
[load\\_vocal\\_audio \(\) \(in module mirdata.ikala\), 36](#)  
[log\\_message \(\) \(in module mirdata.utils\), 66](#)  
[LyricData \(class in mirdata.utils\), 64](#)  
[LyricData \(in module mirdata.utils\), 63](#)  
[lyrics \(mirdata.ikala.Track attribute\), 35](#)  
[lyrics \(mirdata.utils.LyricData attribute\), 64](#)  
[lyrics\\_path \(mirdata.ikala.Track attribute\), 35](#)  
[lyrics\\_to\\_jams \(\) \(in module mirdata.jams\\_utils\), 70](#)

## M

[md5 \(\) \(in module mirdata.utils\), 66](#)  
[melody \(mirdata.orchset.Track attribute\), 47](#)  
[melody1 \(mirdata.medleydb\\_melody.Track attribute\), 41](#)  
[melody1\\_path \(mirdata.medleydb\\_melody.Track attribute\), 40](#)  
[melody2 \(mirdata.medleydb\\_melody.Track attribute\), 41](#)  
[melody2\\_path \(mirdata.medleydb\\_melody.Track attribute\), 40](#)  
[melody3 \(mirdata.medleydb\\_melody.Track attribute\), 41](#)



melody3\_path (*mirdata.medleydb\_melody.Track attribute*), 40  
 melody\_path (*mirdata.orchset.Track attribute*), 47  
 metadata\_path (*mirdata.beatport\_key.Track attribute*), 15  
 metadata\_path (*mirdata.giantsteps\_key.Track attribute*), 23  
 midi (*mirdata.groove\_midi.Track attribute*), 27  
 midi (*mirdata.maestro.Track attribute*), 38  
 midi\_path (*mirdata.groove\_midi.Track attribute*), 27  
 midi\_path (*mirdata.maestro.Track attribute*), 38  
 MIR\_DATASETS\_DIR (*in module mirdata.utils*), 63  
 mirdata (*module*), 12  
 mirdata.beatles (*module*), 12  
 mirdata.beatport\_key (*module*), 14  
 mirdata.dali (*module*), 17  
 mirdata.download\_utils (*module*), 66  
 mirdata.giantsteps\_key (*module*), 23  
 mirdata.giantsteps\_tempo (*module*), 20  
 mirdata.groove\_midi (*module*), 26  
 mirdata.gtzan\_genre (*module*), 29  
 mirdata.guitarset (*module*), 30  
 mirdata.ikala (*module*), 35  
 mirdata.jams\_utils (*module*), 68  
 mirdata.maestro (*module*), 37  
 mirdata.medley\_solos\_db (*module*), 44  
 mirdata.medleydb\_melody (*module*), 40  
 mirdata.medleydb\_pitch (*module*), 42  
 mirdata.orchset (*module*), 46  
 mirdata.rwc\_classical (*module*), 49  
 mirdata.rwc\_jazz (*module*), 51  
 mirdata.rwc\_popular (*module*), 54  
 mirdata.salami (*module*), 56  
 mirdata.tinysol (*module*), 59  
 mirdata.track (*module*), 62  
 mirdata.utils (*module*), 63  
 mix\_audio (*mirdata.ikala.Track attribute*), 36  
 mode (*mirdata.guitarset.Track attribute*), 32  
 multi\_sections\_to\_jams () (*in module mirdata.jams\_utils*), 70  
 MultipitchData (*class in mirdata.utils*), 64  
 MultiTrack (*class in mirdata.track*), 62

## N

n\_sources (*mirdata.medleydb\_melody.Track attribute*), 40  
 none\_path\_join () (*in module mirdata.utils*), 66  
 NoteData (*class in mirdata.utils*), 65  
 NoteData (*in module mirdata.utils*), 63  
 notes (*mirdata.dali.Track attribute*), 19  
 notes (*mirdata.guitarset.Track attribute*), 33  
 notes (*mirdata.maestro.Track attribute*), 38  
 notes (*mirdata.utils.NoteData attribute*), 65  
 notes\_to\_jams () (*in module mirdata.jams\_utils*), 70

## O

only\_brass (*mirdata.orchset.Track attribute*), 47  
 only\_strings (*mirdata.orchset.Track attribute*), 47  
 only\_winds (*mirdata.orchset.Track attribute*), 47

## P

paragraphs (*mirdata.dali.Track attribute*), 19  
 piece\_number (*mirdata.rwc\_classical.Track attribute*), 49  
 piece\_number (*mirdata.rwc\_jazz.Track attribute*), 52  
 piece\_number (*mirdata.rwc\_popular.Track attribute*), 54  
 pitch (*mirdata.medleydb\_pitch.Track attribute*), 43  
 pitch (*mirdata.tinysol.Track attribute*), 60  
 pitch\_contours (*mirdata.guitarset.Track attribute*), 33  
 pitch\_id (*mirdata.tinysol.Track attribute*), 60  
 pitch\_path (*mirdata.medleydb\_pitch.Track attribute*), 42  
 player\_id (*mirdata.guitarset.Track attribute*), 32  
 predominant\_melodic\_instruments (*mirdata.orchset.Track attribute*), 47  
 pronunciations (*mirdata.utils.LyricData attribute*), 64

## R

release\_date (*mirdata.dali.Track attribute*), 18  
 RemoteFileMetadata (*class in mirdata.download\_utils*), 67  
 RemoteFileMetadata (*in module mirdata.download\_utils*), 66

## S

scores\_manual (*mirdata.dali.Track attribute*), 18  
 scores\_ncc (*mirdata.dali.Track attribute*), 18  
 section (*mirdata.ikala.Track attribute*), 35  
 SectionData (*class in mirdata.utils*), 65  
 SectionData (*in module mirdata.utils*), 63  
 sections (*mirdata.beatles.Track attribute*), 13  
 sections (*mirdata.rwc\_classical.Track attribute*), 50  
 sections (*mirdata.rwc\_jazz.Track attribute*), 53  
 sections (*mirdata.rwc\_popular.Track attribute*), 55  
 sections\_annotator1\_lowercase\_path (*mirdata.salami.Track attribute*), 57  
 sections\_annotator1\_uppercase\_path (*mirdata.salami.Track attribute*), 57  
 sections\_annotator2\_lowercase\_path (*mirdata.salami.Track attribute*), 57  
 sections\_annotator2\_uppercase\_path (*mirdata.salami.Track attribute*), 57  
 sections\_annotator1\_lowercase (*mirdata.salami.Track attribute*), 58  
 sections\_annotator1\_uppercase (*mirdata.salami.Track attribute*), 58

sections\_annotator\_2\_lowercase (*mirdata.salami.Track* attribute), 58

sections\_annotator\_2\_uppercase (*mirdata.salami.Track* attribute), 58

sections\_path (*mirdata.beatles.Track* attribute), 12

sections\_path (*mirdata.rwc\_classical.Track* attribute), 49

sections\_path (*mirdata.rwc\_jazz.Track* attribute), 52

sections\_path (*mirdata.rwc\_popular.Track* attribute), 54

sections\_to\_jams() (in module *mirdata.jams\_utils*), 70

session (*mirdata.groove\_midi.Track* attribute), 26

singer\_id (*mirdata.ikala.Track* attribute), 35

singer\_information (*mirdata.rwc\_popular.Track* attribute), 54

song\_id (*mirdata.ikala.Track* attribute), 35

song\_id (*mirdata.medley\_solos\_db.Track* attribute), 44

source (*mirdata.salami.Track* attribute), 57

split (*mirdata.groove\_midi.Track* attribute), 27

split (*mirdata.maestro.Track* attribute), 38

start\_times (*mirdata.utils.EventData* attribute), 64

start\_times (*mirdata.utils.KeyData* attribute), 64

start\_times (*mirdata.utils.LyricData* attribute), 64

string\_id (*mirdata.tinysol.Track* attribute), 60

style (*mirdata.groove\_midi.Track* attribute), 27

style (*mirdata.guitarset.Track* attribute), 32

subset (*mirdata.medley\_solos\_db.Track* attribute), 44

suffix (*mirdata.rwc\_classical.Track* attribute), 50

suffix (*mirdata.rwc\_jazz.Track* attribute), 52

suffix (*mirdata.rwc\_popular.Track* attribute), 55

## T

tag\_gtzan\_to\_jams() (in module *mirdata.jams\_utils*), 71

technique\_abbr (*mirdata.tinysol.Track* attribute), 60

technique\_full (*mirdata.tinysol.Track* attribute), 60

tempo (*mirdata.beatport\_key.Track* attribute), 15

tempo (*mirdata.giantsteps\_key.Track* attribute), 24

tempo (*mirdata.giantsteps\_tempo.Track* attribute), 21

tempo (*mirdata.groove\_midi.Track* attribute), 27

tempo (*mirdata.guitarset.Track* attribute), 32

tempo (*mirdata.rwc\_popular.Track* attribute), 55

tempo\_v2 (*mirdata.giantsteps\_tempo.Track* attribute), 21

TempoData (class in *mirdata.utils*), 65

TempoData (in module *mirdata.utils*), 63

tempos\_to\_jams() (in module *mirdata.jams\_utils*), 71

time (*mirdata.utils.TempoData* attribute), 65

time\_signature (*mirdata.groove\_midi.Track* attribute), 27

times (*mirdata.utils.F0Data* attribute), 64

times (*mirdata.utils.MultipitchData* attribute), 65

title (*mirdata.beatles.Track* attribute), 12

title (*mirdata.beatport\_key.Track* attribute), 15

title (*mirdata.dali.Track* attribute), 18

title (*mirdata.giantsteps\_key.Track* attribute), 24

title (*mirdata.giantsteps\_tempo.Track* attribute), 21

title (*mirdata.medleydb\_melody.Track* attribute), 41

title (*mirdata.medleydb\_pitch.Track* attribute), 42

title (*mirdata.rwc\_classical.Track* attribute), 50

title (*mirdata.rwc\_jazz.Track* attribute), 52

title (*mirdata.rwc\_popular.Track* attribute), 55

title (*mirdata.salami.Track* attribute), 58

to\_jams() (*mirdata.beatles.Track* method), 13

to\_jams() (*mirdata.beatport\_key.Track* method), 16

to\_jams() (*mirdata.dali.Track* method), 19

to\_jams() (*mirdata.giantsteps\_key.Track* method), 24

to\_jams() (*mirdata.giantsteps\_tempo.Track* method), 22

to\_jams() (*mirdata.gtzan\_genre.Track* method), 29

to\_jams() (*mirdata.guitarset.Track* method), 33

to\_jams() (*mirdata.ikala.Track* method), 36

to\_jams() (*mirdata.maestro.Track* method), 38

to\_jams() (*mirdata.medley\_solos\_db.Track* method), 45

to\_jams() (*mirdata.medleydb\_melody.Track* method), 41

to\_jams() (*mirdata.medleydb\_pitch.Track* method), 43

to\_jams() (*mirdata.orchset.Track* method), 47

to\_jams() (*mirdata.rwc\_classical.Track* method), 50

to\_jams() (*mirdata.rwc\_jazz.Track* method), 53

to\_jams() (*mirdata.rwc\_popular.Track* method), 55

to\_jams() (*mirdata.salami.Track* method), 58

to\_jams() (*mirdata.tinysol.Track* method), 61

to\_jams\_v2() (*mirdata.giantsteps\_tempo.Track* method), 22

Track (class in *mirdata.beatles*), 12

Track (class in *mirdata.beatport\_key*), 15

Track (class in *mirdata.dali*), 17

Track (class in *mirdata.giantsteps\_key*), 23

Track (class in *mirdata.giantsteps\_tempo*), 21

Track (class in *mirdata.groove\_midi*), 26

Track (class in *mirdata.gtzan\_genre*), 29

Track (class in *mirdata.guitarset*), 31

Track (class in *mirdata.ikala*), 35

Track (class in *mirdata.maestro*), 37

Track (class in *mirdata.medley\_solos\_db*), 44

Track (class in *mirdata.medleydb\_melody*), 40

Track (class in *mirdata.medleydb\_pitch*), 42

Track (class in *mirdata.orchset*), 46

Track (class in *mirdata.rwc\_classical*), 49

Track (class in *mirdata.rwc\_jazz*), 51

Track (class in *mirdata.rwc\_popular*), 54

Track (class in *mirdata.salami*), 56

Track (class in *mirdata.tinysol*), 59  
*track\_id* (*mirdata.beatles.Track* attribute), 12  
*track\_id* (*mirdata.beatport\_key.Track* attribute), 15  
*track\_id* (*mirdata.dali.Track* attribute), 18  
*track\_id* (*mirdata.giantsteps\_key.Track* attribute), 24  
*track\_id* (*mirdata.giantsteps\_tempo.Track* attribute), 21  
*track\_id* (*mirdata.groove\_midi.Track* attribute), 26  
*track\_id* (*mirdata.gtzan\_genre.Track* attribute), 29  
*track\_id* (*mirdata.guitarset.Track* attribute), 32  
*track\_id* (*mirdata.ikala.Track* attribute), 35  
*track\_id* (*mirdata.maestro.Track* attribute), 38  
*track\_id* (*mirdata.medley\_solos\_db.Track* attribute), 45  
*track\_id* (*mirdata.medleydb\_melody.Track* attribute), 41  
*track\_id* (*mirdata.medleydb\_pitch.Track* attribute), 43  
*track\_id* (*mirdata.orchset.Track* attribute), 47  
*track\_id* (*mirdata.rwc\_classical.Track* attribute), 50  
*track\_id* (*mirdata.rwc\_jazz.Track* attribute), 52  
*track\_id* (*mirdata.rwc\_popular.Track* attribute), 55  
*track\_id* (*mirdata.tinysol.Track* attribute), 60  
*track\_ids* () (in module *mirdata.beatles*), 14  
*track\_ids* () (in module *mirdata.beatport\_key*), 17  
*track\_ids* () (in module *mirdata.dali*), 20  
*track\_ids* () (in module *mirdata.giantsteps\_key*), 25  
*track\_ids* () (in module *mirdata.giantsteps\_tempo*), 22  
*track\_ids* () (in module *mirdata.groove\_midi*), 28  
*track\_ids* () (in module *mirdata.gtzan\_genre*), 30  
*track\_ids* () (in module *mirdata.guitarset*), 34  
*track\_ids* () (in module *mirdata.ikala*), 36  
*track\_ids* () (in module *mirdata.maestro*), 39  
*track\_ids* () (in module *mirdata.medley\_solos\_db*), 45  
*track\_ids* () (in module *mirdata.medleydb\_melody*), 41  
*track\_ids* () (in module *mirdata.medleydb\_pitch*), 43  
*track\_ids* () (in module *mirdata.orchset*), 48  
*track\_ids* () (in module *mirdata.rwc\_classical*), 51  
*track\_ids* () (in module *mirdata.rwc\_jazz*), 53  
*track\_ids* () (in module *mirdata.rwc\_popular*), 56  
*track\_ids* () (in module *mirdata.salami*), 59  
*track\_ids* () (in module *mirdata.tinysol*), 61  
*track\_number* (*mirdata.rwc\_classical.Track* attribute), 50  
*track\_number* (*mirdata.rwc\_jazz.Track* attribute), 52  
*track\_number* (*mirdata.rwc\_popular.Track* attribute), 55

## U

*untar* () (in module *mirdata.download\_utils*), 68  
*unzip* () (in module *mirdata.download\_utils*), 68

*url* (*mirdata.download\_utils.RemoteFileMetadata* attribute), 67  
*url\_working* (*mirdata.dali.Track* attribute), 18

## V

*validate* () (in module *mirdata.beatles*), 14  
*validate* () (in module *mirdata.beatport\_key*), 17  
*validate* () (in module *mirdata.dali*), 20  
*validate* () (in module *mirdata.giantsteps\_key*), 25  
*validate* () (in module *mirdata.giantsteps\_tempo*), 22  
*validate* () (in module *mirdata.groove\_midi*), 29  
*validate* () (in module *mirdata.gtzan\_genre*), 30  
*validate* () (in module *mirdata.guitarset*), 34  
*validate* () (in module *mirdata.ikala*), 37  
*validate* () (in module *mirdata.maestro*), 39  
*validate* () (in module *mirdata.medley\_solos\_db*), 46  
*validate* () (in module *mirdata.medleydb\_melody*), 42  
*validate* () (in module *mirdata.medleydb\_pitch*), 43  
*validate* () (in module *mirdata.orchset*), 48  
*validate* () (in module *mirdata.rwc\_classical*), 51  
*validate* () (in module *mirdata.rwc\_jazz*), 53  
*validate* () (in module *mirdata.rwc\_popular*), 56  
*validate* () (in module *mirdata.salami*), 59  
*validate* () (in module *mirdata.tinysol*), 61  
*validator* () (in module *mirdata.utils*), 66  
*value* (*mirdata.utils.TempoData* attribute), 65  
*variation* (*mirdata.rwc\_jazz.Track* attribute), 52  
*voca\_inst\_path* (*mirdata.rwc\_popular.Track* attribute), 55  
*vocal\_audio* (*mirdata.ikala.Track* attribute), 36  
*vocal\_instrument\_activity* (*mirdata.rwc\_popular.Track* attribute), 55

## W

*words* (*mirdata.dali.Track* attribute), 19  
*work* (*mirdata.orchset.Track* attribute), 47

## Y

*year* (*mirdata.maestro.Track* attribute), 38