

---

# MusPy

Jan 03, 2020



---

## Contents

---

<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Why MusPy</b>	<b>5</b>
<b>3</b>	<b>Installation</b>	<b>7</b>
<b>4</b>	<b>Documentation</b>	<b>9</b>
<b>5</b>	<b>Citing</b>	<b>11</b>
<b>6</b>	<b>Disclaimer</b>	<b>13</b>
<b>7</b>	<b>Contents</b>	<b>15</b>
7.1	Getting Started . . . . .	15
7.2	MusPy Classes . . . . .	17
7.3	Timing in MusPy . . . . .	57
7.4	Input/Output Interfaces . . . . .	57
7.5	Datasets . . . . .	63
7.6	Representations . . . . .	89
7.7	Synthesis . . . . .	98
7.8	Visualization . . . . .	99
7.9	Metrics . . . . .	100
7.10	Technical Documentation . . . . .	106
	<b>Python Module Index</b>	<b>199</b>
	<b>Index</b>	<b>201</b>



MusPy is an open source Python library for symbolic music generation. It provides essential tools for developing a music generation system, including dataset management, data I/O, data preprocessing and model evaluation.



# CHAPTER 1

---

## Features

---

- Dataset management system for commonly used datasets with interfaces to PyTorch and TensorFlow.
- Data I/O for common symbolic music formats (e.g., MIDI, MusicXML and ABC) and interfaces to other symbolic music libraries (e.g., music21, mido, pretty\_midi and Pypianoroll).
- Implementations of common music representations for music generation, including the pitch-based, the event-based, the piano-roll and the note-based representations.
- Model evaluation tools for music generation systems, including audio rendering, score and piano-roll visualizations and objective metrics.

Here is an overview of the library.





## CHAPTER 2

---

### Why MusPy

---

A music generation pipeline usually consists of several steps: data collection, data preprocessing, model creation, model training and model evaluation.

While some components need to be customized for each model, others can be shared across systems. For symbolic music generation in particular, a number of datasets, representations and metrics have been proposed in the literature. As a result, an easy-to-use toolkit that implements standard versions of such routines could save a great deal of time and effort and might lead to increased reproducibility.



## CHAPTER 3

---

### Installation

---

To install MusPy, please run `pip install muspy`. To build MusPy from source, please download the [source](#) and run `python setup.py install`.



## CHAPTER 4

---

### Documentation

---

Documentation is available [here](#) and as docstrings with the code.



## CHAPTER 5

---

### Citing

---

Please cite the following paper if you use MusPy in a published work:

Hao-Wen Dong, Ke Chen, Julian McAuley, and Taylor Berg-Kirkpatrick, “MusPy: A Toolkit for Symbolic Music Generation,” in *Proceedings of the 21st International Society for Music Information Retrieval Conference (ISMIR)*, 2020.

[\[homepage\]](#) [\[video\]](#) [\[paper\]](#) [\[slides\]](#) [\[poster\]](#) [\[arXiv\]](#) [\[code\]](#) [\[documentation\]](#)





## CHAPTER 6

---

### Disclaimer

---

This is a utility library that downloads and prepares public datasets. We do not host or distribute these datasets, vouch for their quality or fairness, or claim that you have license to use the dataset. It is your responsibility to determine whether you have permission to use the dataset under the dataset's license.

If you're a dataset owner and wish to update any part of it (description, citation, etc.), or do not want your dataset to be included in this library, please get in touch through a GitHub issue. Thanks for your contribution to the community!



## 7.1 Getting Started

Welcome to MusPy! We will go through some basic concepts in this tutorial.

---

**Hint:** Be sure you have MusPy installed. To install MusPy, please run `pip install muspy`.

---

In the following example, we will use [this JSON file](#) as an example.

First of all, let's import the MusPy library.

```
import muspy
```

Now, let's load the example JSON file into a Music object.

```
music = muspy.load("example.json")
print(music)
```

Here's what we got.

```
Music(metadata=Metadata(schema_version='0.0', title='Für Elise', creators=['Ludwig_
↳van Beethoven'], collection='Example dataset', source_filename='example.json'),_
↳resolution=4, tempos=[Tempo(time=0, qpm=72.0)], key_signatures=[KeySignature(time=0,
↳root=9, mode='minor')], time_signatures=[TimeSignature(time=0, numerator=3,_
↳denominator=8)], downbeats=[4, 16], lyrics=[Lyric(time=0, lyric='Nothing but a lyric
↳')], annotations=[Annotation(time=0, annotation='Nothing but an annotation')],_
↳tracks=[Track(program=0, is_drum=False, name='Melody', notes=[Note(time=0,_
↳duration=2, pitch=76, velocity=64), Note(time=2, duration=2, pitch=75, velocity=64),
↳Note(time=4, duration=2, pitch=76, velocity=64), ...], lyrics=[Lyric(time=0, lyric=
↳'Nothing but a lyric')], annotations=[Annotation(time=0, annotation='Nothing but an_
↳annotation')]))])
```

Hard to read, isn't it? Let's print it beautifully.

```
music.print()
```

Now here's what we got.

```
metadata:
  schema_version: '0.0'
  title: Für Elise
  creators:
    - Ludwig van Beethoven
  collection: Example dataset
  source_filename: example.json
resolution: 4
tempos:
- time: 0
  qpm: 72.0
key_signatures:
- time: 0
  root: 9
  mode: minor
time_signatures:
- time: 0
  numerator: 3
  denominator: 8
downbeats:
- 4
- 16
lyrics:
- time: 0
  lyric: Nothing but a lyric
annotations:
- time: 0
  annotation: Nothing but an annotation
tracks:
- program: 0
  is_drum: false
  name: Melody
  notes:
    - time: 0
      pitch: 76
      duration: 2
      velocity: 64
    - time: 2
      pitch: 75
      duration: 2
      velocity: 64
    - time: 4
      pitch: 76
      duration: 2
      velocity: 64
    - time: 6
      pitch: 75
      duration: 2
      velocity: 64
    - time: 8
      pitch: 76
      duration: 2
      velocity: 64
```

(continues on next page)

(continued from previous page)

```

- time: 10
  pitch: 71
  duration: 2
  velocity: 64
- time: 12
  pitch: 74
  duration: 2
  velocity: 64
- time: 14
  pitch: 72
  duration: 2
  velocity: 64
- time: 16
  pitch: 69
  duration: 2
  velocity: 64
lyrics:
- time: 0
  lyric: Nothing but a lyric
annotations:
- time: 0
  annotation: Nothing but an annotation

```

You can use dot notation to assess the data. For example, `music.metadata.title` returns the song title, and `music.tempos[0].qpm` returns the first tempo in qpm (quarter notes per minute). If you want a list of all the pitches, you can do

```
print([note.pitch for note in music.tracks[0].notes])
```

Then you will get `[76, 75, 76, 75, 76, 71, 74, 72, 69]`.

---

**Hint:** `music[i]` is a shorthand for `music.tracks[i]`, and `len(music)` for `len(music.tracks)`.

---

There's more MusPy offers. Here is an example of data preparation pipeline using MusPy.

And here is another example of result writing pipeline using MusPy.

## 7.2 MusPy Classes

MusPy provides several classes for working with symbolic music. Here is an illustration of the relations between different MusPy classes.

### 7.2.1 Base Classes

#### Base Class

All MusPy classes inherit from the `muspy.Base` class. A `muspy.Base` object supports the following operations.

- `muspy.Base.to_ordered_dict()`: convert the content into an ordered dictionary
- `muspy.Base.from_dict()` (class method): create a MusPy object of a certain class
- `muspy.Base.print()`: show the content in a YAML-like format
- `muspy.Base.validate()`: validate the data stored in an object
- `muspy.Base.is_valid()`: return a boolean indicating if the stored data is valid
- `muspy.Base.adjust_time()`: adjust the timing of an object

## ComplexBase Class

MusPy classes that contains list attributes also inherit from the `muspy.ComplexBase` class. A `muspy.ComplexBase` object supports the following operations.

- `muspy.ComplexBase.append()`: append an object to the corresponding list
- `muspy.ComplexBase.remove_invalid()`: remove invalid items from the lists
- `muspy.ComplexBase.sort()`: sort the lists
- `muspy.ComplexBase.remove_duplicate()`: remove duplicate items from the lists

## 7.2.2 Music Class

The `muspy.Music` class is the core element of MusPy. It is a universal container for symbolic music.

Attributes	Description	Type	Default
metadata	Metadata	<code>muspy.Metadata</code>	<code>muspy.Metadata()</code>
resolution	Time steps per beat	int	<code>muspy.DEFAULT_RESOLUTION</code>
tempos	Tempo changes	list of <code>muspy.Tempo</code>	<code>[]</code>
key_signatures	Key signature changes	list of <code>muspy.KeySignature</code>	<code>[]</code>
time_signatures	Time signature changes	list of <code>muspy.TimeSignature</code>	<code>[]</code>
downbeats	Downbeat positions	list of int	<code>[]</code>
lyrics	Lyrics	list of <code>muspy.Lyric</code>	<code>[]</code>
annotations	Annotations	list of <code>muspy.Annotation</code>	<code>[]</code>
tracks	Music tracks	list of <code>muspy.Track</code>	<code>[]</code>

---

**Hint:** An example of a MusPy Music object as a YAML file is available [here](#).

---

```
class muspy.Music(metadata: Optional[muspy.classes.Metadata] = None, resolution: Optional[int]
    = None, tempos: Optional[List[muspy.classes.Tempo]] = None, key_signatures:
    Optional[List[muspy.classes.KeySignature]] = None, time_signatures: Op-
    tional[List[muspy.classes.TimeSignature]] = None, downbeats: Optional[List[int]]
    = None, lyrics: Optional[List[muspy.classes.Lyric]] = None, annota-
    tions: Optional[List[muspy.classes.Annotation]] = None, tracks: Op-
    tional[List[muspy.classes.Track]] = None)
```

A universal container for symbolic music.

This is the core class of MusPy. A Music object can be constructed in the following ways.

- `muspy.Music()`: Construct by setting values for attributes.

- `muspy.Music.from_dict()`: Construct from a dictionary that stores the attributes and their values as key-value pairs.
- `muspy.read()`: Read from a MIDI, a MusicXML or an ABC file.
- `muspy.load()`: Load from a JSON or a YAML file saved by `muspy.save()`.
- `muspy.from_object()`: Convert from a `music21.Stream`, a `mido.MidiFile`, a `pretty_midi.PrettyMIDI` or a `pypianoroll.Multitrack` object.

**metadata**

Metadata.

**Type** `muspy.Metadata`**resolution**Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.**Type** `int`, optional**tempos**

Tempo changes.

**Type** list of `muspy.Tempo`**key\_signatures**

Key signatures changes.

**Type** list of `muspy.KeySignature`**time\_signatures**

Time signature changes.

**Type** list of `muspy.TimeSignature`**downbeats**

Downbeat positions.

**Type** list of `int`**lyrics**

Lyrics.

**Type** list of `muspy.Lyric`**annotations**

Annotations.

**Type** list of `muspy.Annotation`**tracks**

Music tracks.

**Type** list of `muspy.Track`


---

**Note:** Indexing a Music object returns the track of a certain index. That is, `music[idx]` returns `music.tracks[idx]`. Length of a Music object is the number of tracks. That is, `len(music)` returns `len(music.tracks)`.

---

**adjust\_resolution** (*target: Optional[int] = None, factor: Optional[float] = None, rounding:*

*Union[str, Callable, None] = 'round')* → `muspy.music.Music`

Adjust resolution and timing of all time-stamped objects.

**Parameters**

- **target** (*int*, *optional*) – Target resolution.
- **factor** (*int* or *float*, *optional*) – Factor used to adjust the resolution based on the formula:  $new\_resolution = old\_resolution * factor$ . For example, a factor of 2 double the resolution, and a factor of 0.5 halve the resolution.
- **rounding** (*{'round', 'ceil', 'floor'}* or *callable*, *optional*) – Rounding mode. Defaults to 'round'.

**Returns**

**Return type** Object itself.

**adjust\_time** (*func: Callable[[int], int]*, *attr: Optional[str] = None*, *recursive: bool = True*) → Base-Type

Adjust the timing of time-stamped objects.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e.,  $new\_time = func(old\_time)$ .
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**append** (*obj*) → ComplexBaseType

Append an object to the corresponding list.

This will automatically determine the list attributes to append based on the type of the object.

**Parameters** *obj* – Object to append.

**clip** (*lower: int = 0*, *upper: int = 127*) → muspy.music.Music

Clip the velocity of each note for each track.

**Parameters**

- **lower** (*int*, *optional*) – Lower bound. Defaults to 0.
- **upper** (*int*, *optional*) – Upper bound. Defaults to 127.

**Returns**

**Return type** Object itself.

**copy** () → BaseType

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

**Returns**

**Return type** Shallow copy of the object.

**deepcopy** () → BaseType

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

**Returns**

**Return type** Deep copy of the object.



**extend** (*other*: Union[ComplexBaseType, Iterable[T\_co]], *deepcopy*: bool = False) → ComplexBaseType  
 Extend the list(s) with another object or iterable.

#### Parameters

- **other** (*muspy.ComplexBase* or iterable) – If an object of the same type is given, extend the list attributes with the corresponding list attributes of the other object. If an iterable is given, call *muspy.ComplexBase.append()* for each item.
- **deepcopy** (*bool*) – Whether to make deep copies of the appended objects. Defaults to False.

#### Returns

**Return type** Object itself.

**classmethod from\_dict** (*dict\_*: Mapping[KT, VT\_co]) → BaseType  
 Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., {"attr1": value1, "attr2": value2}.

#### Returns

**Return type** Constructed object.

**get\_end\_time** (*is\_sorted*: bool = False) → int  
 Return the the time of the last event in all tracks.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations.

**Parameters is\_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

**get\_real\_end\_time** (*is\_sorted*: bool = False) → float  
 Return the end time in realtime.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations. Assume 120 qpm (quarter notes per minute) if no tempo information is available.

**Parameters is\_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

**is\_valid** (*attr*: Optional[str] = None, *recursive*: bool = True) → bool  
 Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** bool

See also:

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

***muspy.Base.validate\_type()*** Raise an error if a certain attribute is of an invalid type.

***muspy.Base.is\_valid()*** Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing: bool = True*) → str

Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** str

See also:

***muspy.Base.print()*** Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

***muspy.Base.pretty\_str()*** Return the the attributes as a string in a YAML-like format.

**remove\_duplicate** (*attr: Optional[str] = None, recursive: bool = True*) → ComplexBaseType

Remove duplicate items from a list attribute.

**Parameters**

- **attr** (*str*) – Attribute to check. Defaults to check all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**remove\_invalid** (*attr: Optional[str] = None, recursive: bool = True*) → ComplexBaseType

Remove invalid items from a list attribute.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.

- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

### Returns

**Return type** Object itself.

**save** (*path: Union[str, pathlib.Path], kind: Optional[str] = None, \*\*kwargs*)

Save loselessly to a JSON or a YAML file.

Refer to [`muspy.save\(\)`](#) for full documentation.

**save\_json** (*path: Union[str, pathlib.Path], \*\*kwargs*)

Save loselessly to a JSON file.

Refer to [`muspy.save\_json\(\)`](#) for full documentation.

**save\_yaml** (*path: Union[str, pathlib.Path]*)

Save loselessly to a YAML file.

Refer to [`muspy.save\_yaml\(\)`](#) for full documentation.

**show** (*kind: str, \*\*kwargs*)

Show visualization.

Refer to [`muspy.show\(\)`](#) for full documentation.

**show\_pianoroll** (*\*\*kwargs*)

Show pianoroll visualization.

Refer to [`muspy.show\_pianoroll\(\)`](#) for full documentation.

**show\_score** (*\*\*kwargs*)

Show score visualization.

Refer to [`muspy.show\_score\(\)`](#) for full documentation.

**sort** (*attr: Optional[str] = None, recursive: bool = True*) → `ComplexBaseType`

Sort a list attribute.

### Parameters

- **attr** (*str*) – Attribute to sort. Defaults to sort all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

### Returns

**Return type** Object itself.

**synthesize** (*\*\*kwargs*) → `numpy.ndarray`

Synthesize a Music object to raw audio.

Refer to [`muspy.synthesize\(\)`](#) for full documentation.

**to\_event\_representation** (*\*\*kwargs*) → `numpy.ndarray`

Return in event-based representation.

Refer to [`muspy.to\_event\_representation\(\)`](#) for full documentation.

**to\_mido** (*\*\*kwargs*) → `mido.midifiles.midifiles.MidiFile`

Return as a `MidiFile` object.

Refer to [`muspy.to\_mido\(\)`](#) for full documentation.

**to\_music21** (*\*\*kwargs*) → `music21.stream.Stream`

Return as a `Stream` object.

Refer to [`muspy.to\_music21\(\)`](#) for full documentation.

**to\_note\_representation** (*\*\*kwargs*) → numpy.ndarray

Return in note-based representation.

Refer to `muspy.to_note_representation()` for full documentation.

**to\_object** (*kind: str, \*\*kwargs*)

Return as an object in other libraries.

Refer to `muspy.to_object()` for full documentation.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → collections.OrderedDict

Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

#### Parameters

- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Return type** OrderedDict

**to\_pianoroll\_representation** (*\*\*kwargs*) → numpy.ndarray

Return in piano-roll representation.

Refer to `muspy.to_pianoroll_representation()` for full documentation.

**to\_pitch\_representation** (*\*\*kwargs*) → numpy.ndarray

Return in pitch-based representation.

Refer to `muspy.to_pitch_representation()` for full documentation.

**to\_pretty\_midi** (*\*\*kwargs*) → pretty\_midi.pretty\_midi.PrettyMIDI

Return as a PrettyMIDI object.

Refer to `muspy.to_pretty_midi()` for full documentation.

**to\_pypianoroll** (*\*\*kwargs*) → pypianoroll.multitrack.Multitrack

Return as a Multitrack object.

Refer to `muspy.to_pypianoroll()` for full documentation.

**to\_representation** (*kind: str, \*\*kwargs*) → numpy.ndarray

Return in a specific representation.

Refer to `muspy.to_representation()` for full documentation.

**transpose** (*semitone: int*) → muspy.music.Music

Transpose all the notes by a number of semitones.

**Parameters** **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

#### Returns

**Return type** Object itself.

## Notes

Drum tracks are skipped.

**validate** (*attr: Optional[str] = None, recursive: bool = True*) → *BaseType*

Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

### Returns

**Return type** Object itself.

See also:

*muspy.Base.is\_valid()* Return True if an attribute has a valid type and value.

*muspy.Base.validate\_type()* Raise an error if an attribute is of an invalid type.

**validate\_type** (*attr: Optional[str] = None, recursive: bool = True*) → *BaseType*

Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

### Returns

**Return type** Object itself.

See also:

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

**write** (*path: Union[str, pathlib.Path], kind: Optional[str] = None, \*\*kwargs*)

Write to a MIDI, a MusicXML, an ABC or an audio file.

Refer to *muspy.write()* for full documentation.

**write\_abc** (*path: Union[str, pathlib.Path], \*\*kwargs*)

Write to an ABC file.

Refer to *muspy.write\_abc()* for full documentation.

**write\_audio** (*path: Union[str, pathlib.Path], \*\*kwargs*)

Write to an audio file.

Refer to *muspy.write\_audio()* for full documentation.

**write\_midi** (*path: Union[str, pathlib.Path], \*\*kwargs*)

Write to a MIDI file.

Refer to *muspy.write\_midi()* for full documentation.

**write\_musicxml** (*path*: Union[str, pathlib.Path], *\*\*kwargs*)

Write to a MusicXML file.

Refer to `muspy.write_musicxml()` for full documentation.

## 7.2.3 Track Class

The `muspy.Track` class is a container for music tracks. In MusPy, each track contains only one instrument.

Attributes	Description	Type	Default
program	MIDI program number	int (0-127)	0
is_drum	If it is a drum track	bool	False
name	Track name	str	
notes	Musical notes	list of <code>muspy.Note</code>	[]
chords	Chords	list of <code>muspy.Chord</code>	[]
lyrics	Lyrics	list of <code>muspy.Lyric</code>	[]
annotations	Annotations	list of <code>muspy.Annotation</code>	[]

(MIDI program number is based on General MIDI specification; see [here](#).)

```
class muspy.Track (program: int = 0, is_drum: bool = False, name: Optional[str] = None, notes: Optional[List[muspy.classes.Note]] = None, chords: Optional[List[muspy.classes.Chord]] = None, lyrics: Optional[List[muspy.classes.Lyric]] = None, annotations: Optional[List[muspy.classes.Annotation]] = None)
```

A container for music track.

### **program**

Program number according to General MIDI specification<sup>1</sup>. Defaults to 0 (Acoustic Grand Piano).

**Type** int, 0-127, optional

### **is\_drum**

Whether it is a percussion track. Defaults to False.

**Type** bool, optional

### **name**

Track name.

**Type** str, optional

### **notes**

Musical notes. Defaults to an empty list.

**Type** list of `muspy.Note`, optional

### **chords**

Chords. Defaults to an empty list.

**Type** list of `muspy.Chord`, optional

### **annotations**

Annotations. Defaults to an empty list.

**Type** list of `muspy.Annotation`, optional

### **lyrics**

Lyrics. Defaults to an empty list.

---

<sup>1</sup> <https://www.midi.org/specifications/item/gm-level-1-sound-set>

**Type** list of *muspy.Lyric*, optional

---

**Note:** Indexing a Track object returns the note at a certain index. That is, `track[idx]` returns `track.notes[idx]`. Length of a Track object is the number of notes. That is, `len(track)` returns `len(track.notes)`.

---

## References

**adjust\_time** (*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → Base-Type

Adjust the timing of time-stamped objects.

### Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., `new_time = func(old_time)`.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

### Returns

**Return type** Object itself.

**append** (*obj*) → ComplexBaseType

Append an object to the corresponding list.

This will automatically determine the list attributes to append based on the type of the object.

**Parameters** *obj* – Object to append.

**clip** (*lower: int = 0, upper: int = 127*) → muspy.classes.Track

Clip the velocity of each note.

### Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

### Returns

**Return type** Object itself.

**copy** () → BaseType

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

### Returns

**Return type** Shallow copy of the object.

**deepcopy** () → BaseType

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

### Returns

**Return type** Deep copy of the object.

**extend** (*other*: Union[ComplexBaseType, Iterable[T\_co]], *deepcopy*: bool = False) → ComplexBaseType

Extend the list(s) with another object or iterable.

**Parameters**

- **other** (*muspy.ComplexBase* or iterable) – If an object of the same type is given, extend the list attributes with the corresponding list attributes of the other object. If an iterable is given, call *muspy.ComplexBase.append()* for each item.
- **deepcopy** (*bool*) – Whether to make deep copies of the appended objects. Defaults to False.

**Returns**

**Return type** Object itself.

**classmethod from\_dict** (*dict\_*: Mapping[KT, VT\_co]) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., {"attr1": value1, "attr2": value2}.

**Returns**

**Return type** Constructed object.

**get\_end\_time** (*is\_sorted*: bool = False) → int

Return the time of the last event.

This includes notes, chords, lyrics and annotations.

**Parameters is\_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

**is\_valid** (*attr*: Optional[str] = None, *recursive*: bool = True) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** bool

**See also:**

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr*: Optional[str] = None, *recursive*: bool = True) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.



**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

***muspy.Base.validate\_type()*** Raise an error if a certain attribute is of an invalid type.

***muspy.Base.is\_valid()*** Return True if an attribute has a valid type and value.

***pretty\_str*** (*skip\_missing: bool = True*) → str

Return the attributes as a string in a YAML-like format.

**Parameters** ***skip\_missing*** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** str

See also:

***muspy.Base.print()*** Print the attributes in a YAML-like format.

***print*** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** ***skip\_missing*** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

***muspy.Base.pretty\_str()*** Return the the attributes as a string in a YAML-like format.

***remove\_duplicate*** (*attr: Optional[str] = None, recursive: bool = True*) → ComplexBaseType

Remove duplicate items from a list attribute.

**Parameters**

- ***attr*** (*str*) – Attribute to check. Defaults to check all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

***remove\_invalid*** (*attr: Optional[str] = None, recursive: bool = True*) → ComplexBaseType

Remove invalid items from a list attribute.

**Parameters**

- ***attr*** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

***sort*** (*attr: Optional[str] = None, recursive: bool = True*) → ComplexBaseType

Sort a list attribute.

**Parameters**

- **attr** (*str*) – Attribute to sort. Defaults to sort all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → `collections.OrderedDict`  
Return the object as an `OrderedDict`.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

**Parameters**

- **skip\_missing** (*bool*) – Whether to skip attributes with value `None` or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Return type** `OrderedDict`

**transpose** (*semitone: int*) → `muspy.classes.Track`  
Transpose the notes by a number of semitones.

**Parameters** **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

**Returns**

**Return type** Object itself.

**validate** (*attr: Optional[str] = None, recursive: bool = True*) → `BaseType`  
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

See also:

`muspy.Base.is_valid()` Return True if an attribute has a valid type and value.

`muspy.Base.validate_type()` Raise an error if an attribute is of an invalid type.

**validate\_type** (*attr: Optional[str] = None, recursive: bool = True*) → `BaseType`  
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**See also:**

`muspy.Base.is_valid_type()` Return True if an attribute is of a valid type.

`muspy.Base.validate()` Raise an error if an attribute has an invalid type or value.

## 7.2.4 Metadata Class

The `muspy.Metadata` class is a container for metadata.

Attributes	Description	Type	Default
<code>schema_version</code>	Schema version	str	'0.0'
<code>title</code>	Song title	str	
<code>creators</code>	Creators(s) of the song	list of str	[]
<code>copyright</code>	Copyright notice	str	
<code>collection</code>	Name of the collection	str	
<code>source_filename</code>	Name of the source file	str	
<code>source_format</code>	Format of the source file	str	

```
class muspy.Metadata (schema_version: str = '0.0', title: Optional[str] = None, creators: Optional[List[str]] = None, copyright: Optional[str] = None, collection: Optional[str] = None, source_filename: Optional[str] = None, source_format: Optional[str] = None)
```

A container for metadata.

**schema\_version**

Schema version. Defaults to the latest version.

**Type** str

**title**

Song title.

**Type** str, optional

**creators**

Creator(s) of the song.

**Type** list of str, optional

**copyright**

Copyright notice.

**Type** str, optional

**collection**

Name of the collection.

**Type** str, optional

**source\_filename**

Name of the source file.

**Type** str, optional

**source\_format**

Format of the source file.

**Type** str, optional

**adjust\_time** (*func*: Callable[[int], int], *attr*: Optional[str] = None, *recursive*: bool = True) → BaseType

Adjust the timing of time-stamped objects.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new\_time* = *func*(*old\_time*).
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**copy** () → BaseType

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

**Returns**

**Return type** Shallow copy of the object.

**deepcopy** () → BaseType

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

**Returns**

**Return type** Deep copy of the object.

**classmethod from\_dict** (*dict\_*: Mapping[KT, VT\_co]) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., {"attr1": value1, "attr2": value2}.

**Returns**

**Return type** Constructed object.

**is\_valid** (*attr*: Optional[str] = None, *recursive*: bool = True) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** bool

**See also:**

[`muspy.Base.validate\(\)`](#) Raise an error if an attribute has an invalid type or value.

[`muspy.Base.is\_valid\_type\(\)`](#) Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

**muspy.Base.validate\_type()** Raise an error if a certain attribute is of an invalid type.

**muspy.Base.is\_valid()** Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing: bool = True*) → str

Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** str

See also:

**muspy.Base.print()** Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

**muspy.Base.pretty\_str()** Return the the attributes as a string in a YAML-like format.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → collections.OrderedDict

Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

#### Parameters

- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., {"attr1": value1, "attr2": value2}.

**Return type** OrderedDict

**validate** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType

Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

**Return type** Object itself.

See also:

*muspy.Base.is\_valid()* Return True if an attribute has a valid type and value.

*muspy.Base.validate\_type()* Raise an error if an attribute is of an invalid type.

**validate\_type** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType

Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

**Return type** Object itself.

See also:

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

## 7.2.5 Tempo Class

The *muspy.Tempo* class is a container for tempos.

Attributes	Description	Type	Default
time	Start time of the tempo	int	
qpm	Tempo in qpm (quarter notes per minute)	float	

**class** *muspy.Tempo* (*time: int, qpm: float*)

A container for key signature.

#### **time**

Start time of the tempo, in time steps.

**Type** *int*

#### **qpm**

Tempo in qpm (quarters per minute).

**Type** *float*

**adjust\_time** (*func*: Callable[[int], int], *attr*: Optional[str] = None, *recursive*: bool = True) → BaseType

Adjust the timing of time-stamped objects.

#### Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new\_time* = *func*(*old\_time*).
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

**Return type** Object itself.

**copy** () → BaseType

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

#### Returns

**Return type** Shallow copy of the object.

**deepcopy** () → BaseType

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

#### Returns

**Return type** Deep copy of the object.

**classmethod from\_dict** (*dict\_*: Mapping[KT, VT\_co]) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., {"attr1": value1, "attr2": value2}.

#### Returns

**Return type** Constructed object.

**is\_valid** (*attr*: Optional[str] = None, *recursive*: bool = True) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** bool

See also:

`muspy.Base.validate()` Raise an error if an attribute has an invalid type or value.

`muspy.Base.is_valid_type()` Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

***muspy.Base.validate\_type()*** Raise an error if a certain attribute is of an invalid type.

***muspy.Base.is\_valid()*** Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing: bool = True*) → str

Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** str

See also:

***muspy.Base.print()*** Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

***muspy.Base.pretty\_str()*** Return the the attributes as a string in a YAML-like format.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → collections.OrderedDict

Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

**Parameters**

- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., {"attr1": value1, "attr2": value2}.

**Return type** OrderedDict



**validate** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType

Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

**Return type** Object itself.

See also:

*muspy.Base.is\_valid()* Return True if an attribute has a valid type and value.

*muspy.Base.validate\_type()* Raise an error if an attribute is of an invalid type.

**validate\_type** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType

Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

**Return type** Object itself.

See also:

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

## 7.2.6 KeySignature Class

The *muspy.KeySignature* class is a container for key signatures.

Attributes	Description	Type	Default
time	Start time	int	
root	Root note as a number	int	
mode	Mode (e.g., “major”)	str	
root_str	Root note as a string	int	

**class** *muspy.KeySignature* (*time: int, root: Optional[int] = None, mode: Optional[str] = None, fifths: Optional[int] = None, root\_str: Optional[str] = None*)

A container for key signature.

**time**

Start time of the key signature, in time steps or seconds.

**Type** *int*

**root**

Root (tonic) of the key signature.

**Type** `int`, optional

**mode**

Mode of the key signature.

**Type** `str`, optional

**fifths**

Number of sharps or flats. Positive numbers for sharps and negative numbers for flats.

**Type** `int`, optional

**root\_str**

Root of the key signature as a string.

**Type** `str`, optional

**adjust\_time** (*func*: Callable[[int], int], *attr*: Optional[str] = None, *recursive*: bool = True) → BaseType

Adjust the timing of time-stamped objects.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new\_time* = *func*(*old\_time*).
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**copy** () → BaseType

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

**Returns**

**Return type** Shallow copy of the object.

**deepcopy** () → BaseType

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

**Returns**

**Return type** Deep copy of the object.

**classmethod from\_dict** (*dict\_*: Mapping[KT, VT\_co]) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters** **dict** (*dict* or *mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., {"attr1": value1, "attr2": value2}.

**Returns**

**Return type** Constructed object.

**is\_valid** (*attr*: Optional[str] = None, *recursive*: bool = True) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** *bool*

**See also:**

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr: Optional[str] = None, recursive: bool = True*) → *bool*

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**See also:**

*muspy.Base.validate\_type()* Raise an error if a certain attribute is of an invalid type.

*muspy.Base.is\_valid()* Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing: bool = True*) → *str*

Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** *str*

**See also:**

*muspy.Base.print()* Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**See also:**

*muspy.Base.pretty\_str()* Return the the attributes as a string in a YAML-like format.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → *collections.OrderedDict*

Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

**Parameters**

- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Return type** OrderedDict

**validate** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType  
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**See also:**

`muspy.Base.is_valid()` Return True if an attribute has a valid type and value.

`muspy.Base.validate_type()` Raise an error if an attribute is of an invalid type.

**validate\_type** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType  
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**See also:**

`muspy.Base.is_valid_type()` Return True if an attribute is of a valid type.

`muspy.Base.validate()` Raise an error if an attribute has an invalid type or value.

## 7.2.7 TimeSignature Class

The `muspy.TimeSignature` class is a container for time signatures.

Attributes	Description	Type	Default
time	Start time	int	
numerator	Numerator (e.g., “3” for 3/4)	int	
denominator	Denominator (e.g., “4” for 3/4)	int	

**class** `muspy.TimeSignature` (*time: int, numerator: int, denominator: int*)

A container for time signature.

**time**

Start time of the time signature, in time steps or seconds.

**Type** `int`

**numerator**

Numerator of the time signature.

**Type** `int`

**denominator**

Denominator of the time signature.

**Type** `int`

**adjust\_time** (*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → `BaseType`

Adjust the timing of time-stamped objects.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., `new_time = func(old_time)`.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**copy** () → `BaseType`

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

**Returns**

**Return type** Shallow copy of the object.

**deepcopy** () → `BaseType`

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

**Returns**

**Return type** Deep copy of the object.

**classmethod from\_dict** (*dict\_: Mapping[KT, VT\_co]*) → `BaseType`

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters** **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Returns**

**Return type** Constructed object.

**is\_valid** (*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** bool

See also:

***muspy.Base.validate()*** Raise an error if an attribute has an invalid type or value.

***muspy.Base.is\_valid\_type()*** Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

***muspy.Base.validate\_type()*** Raise an error if a certain attribute is of an invalid type.

***muspy.Base.is\_valid()*** Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing: bool = True*) → str

Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** str

See also:

***muspy.Base.print()*** Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

***muspy.Base.pretty\_str()*** Return the the attributes as a string in a YAML-like format.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → collections.OrderedDict  
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

#### Parameters

- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., {“attr1”: value1, “attr2”: value2}.

**Return type** OrderedDict

**validate** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType  
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute’s attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

**Return type** Object itself.

See also:

**muspy.Base.is\_valid()** Return True if an attribute has a valid type and value.

**muspy.Base.validate\_type()** Raise an error if an attribute is of an invalid type.

**validate\_type** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType  
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute’s attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

**Return type** Object itself.

See also:

**muspy.Base.is\_valid\_type()** Return True if an attribute is of a valid type.

**muspy.Base.validate()** Raise an error if an attribute has an invalid type or value.

## 7.2.8 Lyric Class

The `muspy.Lyric` class is a container for lyrics.

Attributes	Description	Type	Default
time	Start time	int	
lyric	Lyric (sentence, word, syllable, etc.)	str	

**class** `muspy.Lyric` (*time: int, lyric: str*)

A container for lyric.

**time**

Start time of the lyric, in time steps or seconds.

**Type** `int`

**lyric**

Lyric (sentence, word, syllable, etc.).

**Type** `str`

**adjust\_time** (*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*)  $\rightarrow$  BaseType

Adjust the timing of time-stamped objects.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new\_time = func(old\_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**copy** ()  $\rightarrow$  BaseType

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

**Returns**

**Return type** Shallow copy of the object.

**deepcopy** ()  $\rightarrow$  BaseType

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

**Returns**

**Return type** Deep copy of the object.

**classmethod from\_dict** (*dict\_: Mapping[KT, VT\_co]*)  $\rightarrow$  BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters** **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Returns**

**Return type** Constructed object.



**is\_valid** (*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** bool

See also:

***muspy.Base.validate()*** Raise an error if an attribute has an invalid type or value.

***muspy.Base.is\_valid\_type()*** Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

***muspy.Base.validate\_type()*** Raise an error if a certain attribute is of an invalid type.

***muspy.Base.is\_valid()*** Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing: bool = True*) → str

Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** str

See also:

***muspy.Base.print()*** Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

***muspy.Base.pretty\_str()*** Return the the attributes as a string in a YAML-like format.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → collections.OrderedDict  
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

**Parameters**

- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., {“attr1”: value1, “attr2”: value2}.

**Return type** OrderedDict

**validate** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType  
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

See also:

*muspy.Base.is\_valid()* Return True if an attribute has a valid type and value.

*muspy.Base.validate\_type()* Raise an error if an attribute is of an invalid type.

**validate\_type** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType  
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

See also:

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

## 7.2.9 Annotation Class

The *muspy.Annotation* class is a container for annotations. For flexibility, *annotation* can hold any type of data.

Attributes	Description	Type	Default
time	Start time	int	
annotation	Annotation of any type		

**class** `muspy.Annotation` (*time: int, annotation: Any, group: Optional[str] = None*)

A container for annotation.

**time**

Start time of the annotation, in time steps or seconds.

**Type** `int`

**annotation**

Annotation of any type.

**Type** `any`

**group**

Group name for better organizing the annotations.

**Type** `str`, optional

**adjust\_time** (*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → `BaseType`

Adjust the timing of time-stamped objects.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., `new_time = func(old_time)`.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to `True`.

**Returns**

**Return type** Object itself.

**copy** () → `BaseType`

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

**Returns**

**Return type** Shallow copy of the object.

**deepcopy** () → `BaseType`

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

**Returns**

**Return type** Deep copy of the object.

**classmethod from\_dict** (*dict\_: Mapping[KT, VT\_co]*) → `BaseType`

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters** **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Returns**

**Return type** Constructed object.

**is\_valid** (*attr: Optional[str] = None, recursive: bool = True*) → bool  
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** bool

**See also:**

***muspy.Base.validate()*** Raise an error if an attribute has an invalid type or value.

***muspy.Base.is\_valid\_type()*** Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr: Optional[str] = None, recursive: bool = True*) → bool  
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**See also:**

***muspy.Base.validate\_type()*** Raise an error if a certain attribute is of an invalid type.

***muspy.Base.is\_valid()*** Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing: bool = True*) → str  
Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** str

**See also:**

***muspy.Base.print()*** Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)  
Print the attributes in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**See also:**

**`muspy.Base.pretty_str()`** Return the the attributes as a string in a YAML-like format.

**`to_ordered_dict`** (*skip\_missing: bool = True, deepcopy: bool = True*) → `collections.OrderedDict`  
Return the object as an `OrderedDict`.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

#### Parameters

- **`skip_missing`** (*bool*) – Whether to skip attributes with value `None` or those that are empty lists. Defaults to `True`.
- **`deepcopy`** (*bool*) – Whether to make deep copies of the attributes. Defaults to `True`.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Return type** `OrderedDict`

**`validate`** (*attr: Optional[str] = None, recursive: bool = True*) → `BaseType`

Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute’s attributes.

#### Parameters

- **`attr`** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **`recursive`** (*bool*) – Whether to apply recursively. Defaults to `True`.

#### Returns

**Return type** Object itself.

See also:

**`muspy.Base.is_valid()`** Return `True` if an attribute has a valid type and value.

**`muspy.Base.validate_type()`** Raise an error if an attribute is of an invalid type.

**`validate_type`** (*attr: Optional[str] = None, recursive: bool = True*) → `BaseType`

Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute’s attributes.

#### Parameters

- **`attr`** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **`recursive`** (*bool*) – Whether to apply recursively. Defaults to `True`.

#### Returns

**Return type** Object itself.

See also:

**`muspy.Base.is_valid_type()`** Return `True` if an attribute is of a valid type.

**`muspy.Base.validate()`** Raise an error if an attribute has an invalid type or value.

## 7.2.10 Note Class

The `muspy.Note` class is a container for musical notes.

Attributes	Description	Type	Default
<code>time</code>	Start time	<code>int</code>	
<code>duration</code>	Note duration, in time steps	<code>int</code>	
<code>pitch</code>	Note pitch as a MIDI note number	<code>int (0-127)</code>	
<code>velocity</code>	Note velocity	<code>int (0-127)</code>	

---

**Hint:** `muspy.Note` has a property `end` with setter and getter implemented, which can be handy sometimes.

---

```
class muspy.Note (time: int, pitch: int, duration: int, velocity: Optional[int] = None, pitch_str: Op-
                    tional[str] = None)
```

A container for note.

**time**

Start time of the note, in time steps.

**Type** `int`

**pitch**

Note pitch, as a MIDI note number.

**Type** `int`

**duration**

Duration of the note, in time steps.

**Type** `int`

**velocity**

Note velocity. Defaults to `muspy.DEFAULT_VELOCITY`.

**Type** `int`, optional

**pitch\_str**

Note pitch as a string, useful for distinguishing C# and Db.

**Type** `str`

```
adjust_time (func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True) →
              muspy.classes.Note
```

Adjust the timing of the note.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., `new_time = func(old_time)`.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

```
clip (lower: int = 0, upper: int = 127) → muspy.classes.Note
```

Clip the velocity of the note.

**Parameters**

- **lower** (*int*, *optional*) – Lower bound. Defaults to 0.
- **upper** (*int*, *optional*) – Upper bound. Defaults to 127.

**Returns**

**Return type** Object itself.

**copy** () → BaseType

Return a shallow copy of the object.

This is equivalent to `copy.copy(self)`.

**Returns**

**Return type** Shallow copy of the object.

**deepcopy** () → BaseType

Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)`.

**Returns**

**Return type** Deep copy of the object.

**end**

End time of the note.

**classmethod from\_dict** (*dict\_*: *Mapping[KT, VT\_co]*) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters** **dict** (*dict* or *mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Returns**

**Return type** Constructed object.

**is\_valid** (*attr*: *Optional[str]* = *None*, *recursive*: *bool* = *True*) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** bool

See also:

**`muspy.Base.validate()`** Raise an error if an attribute has an invalid type or value.

**`muspy.Base.is_valid_type()`** Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr*: *Optional[str]* = *None*, *recursive*: *bool* = *True*) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**See also:**

***muspy.Base.validate\_type()*** Raise an error if a certain attribute is of an invalid type.

***muspy.Base.is\_valid()*** Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing: bool = True*) → *str*

Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** *str*

**See also:**

***muspy.Base.print()*** Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**See also:**

***muspy.Base.pretty\_str()*** Return the the attributes as a string in a YAML-like format.

**start**

Start time of the note.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → *collections.OrderedDict*

Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

**Parameters**

- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., *{“attr1”: value1, “attr2”: value2}*.

**Return type** *OrderedDict*

**transpose** (*semitone: int*) → *muspy.classes.Note*

Transpose the note by a number of semitones.

**Parameters** **semitone** (*int*) – Number of semitones to transpose the note. A positive value raises the pitch, while a negative value lowers the pitch.



**Returns****Return type** Object itself.**validate** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType

Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns****Return type** Object itself.**See also:***muspy.Base.is\_valid()* Return True if an attribute has a valid type and value.*muspy.Base.validate\_type()* Raise an error if an attribute is of an invalid type.**validate\_type** (*attr: Optional[str] = None, recursive: bool = True*) → BaseType

Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns****Return type** Object itself.**See also:***muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

## 7.2.11 Chord Class

The *muspy.Chord* class is a container for chords.

Attributes	Description	Type	Default
time	Start time	int	
duration	Chord duration, in time steps	int	
pitch	Note pitches as MIDI note numbers	list of int (0-127)	[]
velocity	Chord velocity	int (0-127)	

**Hint:** *muspy.Chord* has a property *end* with setter and getter implemented, which can be handy sometimes.

```
class muspy.Chord(time: int, pitches: List[int], duration: int, velocity: Optional[int] = None,  
                  pitches_str: Optional[List[int]] = None)
```

A container for chord.

**time**  
Start time of the chord, in time steps.  
**Type** `int`

**pitches**  
Note pitches, as MIDI note numbers.  
**Type** `list of int`

**duration**  
Duration of the chord, in time steps.  
**Type** `int`

**velocity**  
Chord velocity. Defaults to `muspy.DEFAULT_VELOCITY`.  
**Type** `int`, optional

**pitches\_str**  
Note pitches as strings, useful for distinguishing C# and Db.  
**Type** `list of str`

**adjust\_time** (*func*: `Callable[[int], int]`, *attr*: `Optional[str] = None`, *recursive*: `bool = True`) → `muspy.classes.Chord`  
Adjust the timing of the chord.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., `new_time = func(old_time)`.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to `True`.

**Returns**  
**Return type** `Object` itself.

**clip** (*lower*: `int = 0`, *upper*: `int = 127`) → `muspy.classes.Chord`  
Clip the velocity of the chord.

**Parameters**

- **lower** (*int*, *optional*) – Lower bound. Defaults to 0.
- **upper** (*int*, *optional*) – Upper bound. Defaults to 127.

**Returns**  
**Return type** `Object` itself.

**copy** () → `BaseType`  
Return a shallow copy of the object.  
This is equivalent to `copy.copy(self)()`.

**Returns**  
**Return type** `Shallow copy of the object`.

**deepcopy** () → `BaseType`  
Return a deep copy of the object.  
This is equivalent to `copy.deepcopy(self)()`

**Returns**

**Return type** Deep copy of the object.

**end**

End time of the chord.

**classmethod from\_dict** (*dict\_*: *Mapping*[*KT*, *VT\_co*]) → *BaseType*

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters** **dict** (*dict* or *mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., {"attr1": value1, "attr2": value2}.

**Returns**

**Return type** Constructed object.

**is\_valid** (*attr*: *Optional*[*str*] = *None*, *recursive*: *bool* = *True*) → *bool*

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** *bool*

See also:

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr*: *Optional*[*str*] = *None*, *recursive*: *bool* = *True*) → *bool*

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

*muspy.Base.validate\_type()* Raise an error if a certain attribute is of an invalid type.

*muspy.Base.is\_valid()* Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing*: *bool* = *True*) → *str*

Return the attributes as a string in a YAML-like format.

**Parameters** **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** `str`

**See also:**

`muspy.Base.print()` Print the attributes in a YAML-like format.

**print** (*skip\_missing: bool = True*)

Print the attributes in a YAML-like format.

**Parameters** `skip_missing (bool)` – Whether to skip attributes with value `None` or those that are empty lists. Defaults to `True`.

**See also:**

`muspy.Base.pretty_str()` Return the the attributes as a string in a YAML-like format.

**start**

Start time of the chord.

**to\_ordered\_dict** (*skip\_missing: bool = True, deepcopy: bool = True*) → `collections.OrderedDict`

Return the object as an `OrderedDict`.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

**Parameters**

- **skip\_missing** (*bool*) – Whether to skip attributes with value `None` or those that are empty lists. Defaults to `True`.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to `True`.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Return type** `OrderedDict`

**transpose** (*semitone: int*) → `muspy.classes.Chord`

Transpose the notes by a number of semitones.

**Parameters** **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

**Returns**

**Return type** Object itself.

**validate** (*attr: Optional[str] = None, recursive: bool = True*) → `BaseType`

Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to `True`.

**Returns**

**Return type** Object itself.

**See also:**

`muspy.Base.is_valid()` Return `True` if an attribute has a valid type and value.

`muspy.Base.validate_type()` Raise an error if an attribute is of an invalid type.

`validate_type(attr: Optional[str] = None, recursive: bool = True) → BaseType`  
 Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

#### Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

#### Returns

**Return type** Object itself.

See also:

`muspy.Base.is_valid_type()` Return True if an attribute is of a valid type.

`muspy.Base.validate()` Raise an error if an attribute has an invalid type or value.

## 7.3 Timing in MusPy

In MusPy, the *metrical timing* is used. That is, time is stored in musically-meaningful unit (e.g., beats, quarter notes). For playback ability, additional resolution and tempo information is needed.

In a metrical timing system, the smallest unit of time is a factor of a beat, which depends on the time signatures and is set to a quarter note by default. We will refer to this smallest unit of time as a *time step*.

Here is the formula relating the metrical and the absolute timing systems.

$$absolute\_time = \frac{60 \times tempo}{resolution} \times metrical\_time$$

Here, *resolution* is the number of time steps per beat and *tempo* is the current tempo (in quarters per minute, or qpm). These two values are stored in a `muspy.Music` object as attributes `music.resolution` and `music.tempos`.

The following are some illustrations of the relationships between time steps and time.

When reading a MIDI file, `music.resolution` is set to the pulses per quarter note (a.k.a., PPQ, PPQN, ticks per beat). When reading a MusicXML file, `music.resolution` is set to the *division* attribute, which determines the number of divisions per quarter note. When multiple division attributes are found, `music.resolution` is set to the least common multiple of them.

## 7.4 Input/Output Interfaces

MusPy provides three type of data I/O interfaces.

- Common symbolic music formats: `muspy.read_*` and `muspy.write_*`
- MusPy's native JSON and YAML formats: `muspy.load_*` and `muspy.save_*`
- Other symbolic music libraries: `muspy.from_*` and `muspy.to_*`

### 7.4.1 MIDI I/O Interface

`muspy.read_midi(path: Union[str, pathlib.Path], backend: str = 'mido', duplicate_note_mode: str = 'fifo') → muspy.music.Music`

Read a MIDI file into a Music object.

#### Parameters

- **path** (*str* or *Path*) – Path to the MIDI file to read.
- **backend** ({'mido', 'pretty\_midi'}) – Backend to use.
- **duplicate\_note\_mode** ({'fifo', 'lifo', 'close\_all'}) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspondng note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'. Only used when *backend*='mido'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out):close the latest note on
  - 'close\_all': close all note on messages

**Returns** Converted Music object.

**Return type** *muspy.Music*

`muspy.write_midi(path: Union[str, pathlib.Path], music: Music, backend: str = 'mido', **kwargs)`

Write a Music object to a MIDI file.

#### Parameters

- **path** (*str* or *Path*) – Path to write the MIDI file.
- **music** (*muspy.Music*) – Music object to write.
- **backend** ({'mido', 'pretty\_midi'}, *optional*) – Backend to use. Defaults to 'mido'.

See also:

`write_midi_mido()` Write a Music object to a MIDI file using mido as backend.

`write_midi_pretty_midi()` Write a Music object to a MIDI file using pretty\_midi as backend.

### 7.4.2 MusicXML Interface

`muspy.read_musicxml(path: Union[str, pathlib.Path], compressed: Optional[bool] = None) → muspy.music.Music`

Read a MusicXML file into a Music object.

**Parameters** **path** (*str* or *Path*) – Path to the MusicXML file to read.

**Returns** Converted Music object.

**Return type** *muspy.Music*

#### Notes

Grace notes and unpitched notes are not supported.

`muspy.write_musicxml` (*path*: Union[str, pathlib.Path], *music*: Music, *compressed*: Optional[bool] = None)

Write a Music object to a MusicXML file.

#### Parameters

- **path** (*str* or *Path*) – Path to write the MusicXML file.
- **music** (*muspy.Music*) – Music object to write.
- **compressed** (*bool*, *optional*) – Whether to write to a compressed MusicXML file. If None, infer from the extension of the filename (‘.xml’ and ‘.musicxml’ for an uncompressed file, ‘.mxl’ for a compressed file).

### 7.4.3 ABC Interface

`muspy.read_abc` (*path*: Union[str, pathlib.Path], *number*: Optional[int] = None, *resolution*=24) → Union[muspy.music.Music, List[muspy.music.Music]]

Return an ABC file into Music object(s) using music21 backend.

#### Parameters

- **path** (*str* or *Path*) – Path to the ABC file to read.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the ‘X:’ field).
- **resolution** (*int*, *optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.

**Returns** Converted Music object(s).

**Return type** list of *muspy.Music*

`muspy.write_abc` (*path*: Union[str, pathlib.Path], *music*: Music)

Write a Music object to a ABC file.

#### Parameters

- **path** (*str* or *Path*) – Path to write the ABC file.
- **music** (*muspy.Music*) – Music object to write.

### 7.4.4 JSON Interface

`muspy.load_json` (*path*: Union[str, pathlib.Path, TextIO], *compressed*: Optional[bool] = None) → *muspy.music.Music*

Load a JSON file into a Music object.

#### Parameters

- **path** (*str*, *Path* or *TextIO*) – Path to the file or the file to load.
- **compressed** (*bool*, *optional*) – Whether the file is a compressed JSON file (*.json.gz*). Has no effect when *path* is a file object. Defaults to infer from the extension (*.gz*).

**Returns** Loaded Music object.

**Return type** *muspy.Music*

## Notes

When a path is given, assume UTF-8 encoding and gzip compression if *compressed=True*.

`muspy.save_json` (*path*: Union[str, pathlib.Path, TextIO], *music*: Music, *skip\_missing*: bool = True, *ensure\_ascii*: bool = False, *compressed*: Optional[bool] = None, *\*\*kwargs*)  
Save a Music object to a JSON file.

### Parameters

- **path** (*str*, Path or TextIO) – Path or file to save the JSON data.
- **music** (*muspy.Music*) – Music object to save.
- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **ensure\_ascii** (*bool*) – Whether to escape non-ASCII characters. Will be passed to PyYAML’s *yaml.dump*. Defaults to False.
- **compressed** (*bool*, optional) – Whether to save as a compressed JSON file (*.json.gz*). Has no effect when *path* is a file object. Defaults to infer from the extension (*.gz*).
- **\*\*kwargs** – Keyword arguments to pass to `json.dumps()`.

## Notes

When a path is given, use UTF-8 encoding and gzip compression if *compressed=True*.

---

**Note:** A [JSON schema](#) is available for validating a JSON file against MusPy’s format.

---

## 7.4.5 YAML Interface

`muspy.load_json` (*path*: Union[str, pathlib.Path, TextIO], *compressed*: Optional[bool] = None) → *muspy.music.Music*  
Load a JSON file into a Music object.

### Parameters

- **path** (*str*, Path or TextIO) – Path to the file or the file to load.
- **compressed** (*bool*, optional) – Whether the file is a compressed JSON file (*.json.gz*). Has no effect when *path* is a file object. Defaults to infer from the extension (*.gz*).

**Returns** Loaded Music object.

**Return type** *muspy.Music*

## Notes

When a path is given, assume UTF-8 encoding and gzip compression if *compressed=True*.

`muspy.save_json` (*path*: Union[str, pathlib.Path, TextIO], *music*: Music, *skip\_missing*: bool = True, *ensure\_ascii*: bool = False, *compressed*: Optional[bool] = None, *\*\*kwargs*)  
Save a Music object to a JSON file.



### Parameters

- **path** (*str*, *Path* or *TextIO*) – Path or file to save the JSON data.
- **music** (*muspy.Music*) – Music object to save.
- **skip\_missing** (*bool*) – Whether to skip attributes with value *None* or those that are empty lists. Defaults to *True*.
- **ensure\_ascii** (*bool*) – Whether to escape non-ASCII characters. Will be passed to PyYAML’s *yaml.dump*. Defaults to *False*.
- **compressed** (*bool*, *optional*) – Whether to save as a compressed JSON file (*.json.gz*). Has no effect when *path* is a file object. Defaults to infer from the extension (*.gz*).
- **\*\*kwargs** – Keyword arguments to pass to *json.dumps()*.

### Notes

When a path is given, use UTF-8 encoding and gzip compression if *compressed=True*.

---

**Note:** A [YAML schema](#) is available for validating a YAML file against MusPy’s format.

---

## 7.4.6 Mido Interface

*muspy.from\_mido* (*midi*: *mido.midfiles.midfiles.MidiFile*, *duplicate\_note\_mode*: *str* = *'fifo'*) → *muspy.music.Music*  
 Return a mido *MidiFile* object as a *Music* object.

### Parameters

- **midi** (*mido.MidiFile*) – Mido *MidiFile* object to convert.
- **duplicate\_note\_mode** (*{'fifo', 'lifo', 'close\_all'}*) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple corresponding note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to *'fifo'*.
  - *'fifo'* (first in first out): close the earliest note on
  - *'lifo'* (first in first out): close the latest note on
  - *'close\_all'*: close all note on messages

**Returns** Converted *Music* object.

**Return type** *muspy.Music*

*muspy.to\_mido* (*music*: *Music*, *use\_note\_off\_message*: *bool* = *False*)  
 Return a *Music* object as a *MidiFile* object.

### Parameters

- **music** (*muspy.Music* object) – Music object to convert.
- **use\_note\_off\_message** (*bool*, *optional*) – Whether to use note-off messages. If *False*, note-on messages with zero velocity are used instead. The advantage to using note-on messages at zero velocity is that it can avoid sending additional status bytes when Running Status is employed. Defaults to *False*.

**Returns** Converted MidiFile object.

**Return type** `mido.MidiFile`

## 7.4.7 music21 Interface

`muspy.from_music21` (*stream: music21.stream.Stream, resolution=24*) → Union[muspy.music.Music, List[muspy.music.Music], muspy.classes.Track, List[muspy.classes.Track]]  
Return a music21 Stream object as Music or Track object(s).

### Parameters

- **stream** (*music21.stream.Stream*) – Stream object to convert.
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.

**Returns** Converted Music or Track object(s).

**Return type** *muspy.Music* or *muspy.Track*

`muspy.to_music21` (*music: Music*) → *music21.stream.Score*  
Return a Music object as a music21 Score object.

**Parameters** **music** (*muspy.Music*) – Music object to convert.

**Returns** Converted music21 Score object.

**Return type** *music21.stream.Score*

## 7.4.8 pretty\_midi Interface

`muspy.from_pretty_midi` (*midi: pretty\_midi.pretty\_midi.PrettyMIDI, resolution: Optional[int] = None*) → *muspy.music.Music*  
Return a pretty\_midi PrettyMIDI object as a Music object.

### Parameters

- **midi** (*pretty\_midi.PrettyMIDI*) – PrettyMIDI object to convert.
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.

**Returns** Converted Music object.

**Return type** *muspy.Music*

`muspy.to_pretty_midi` (*music: Music*) → *pretty\_midi.pretty\_midi.PrettyMIDI*  
Return a Music object as a PrettyMIDI object.

Tempo changes are not supported yet.

**Parameters** **music** (*muspy.Music* object) – Music object to convert.

**Returns** Converted PrettyMIDI object.

**Return type** *pretty\_midi.PrettyMIDI*

### Notes

Tempo information will not be included in the output.

## 7.4.9 Pypianoroll Interface

`muspy.from_pypianoroll` (*multitrack*: `pypianoroll.multitrack.Multitrack`, *default\_velocity*: `int = 64`) → `muspy.music.Music`

Return a Pypianoroll Multitrack object as a Music object.

### Parameters

- **multitrack** (`pypianoroll.Multitrack`) – Pypianoroll Multitrack object to convert.
- **default\_velocity** (`int`) – Default velocity value to use when decoding. Defaults to 64.

**Returns** `music` – Converted MusPy Music object.

**Return type** `muspy.Music`

`muspy.to_pypianoroll` (*music*: `Music`) → `pypianoroll.multitrack.Multitrack`

Return a Music object as a Multitrack object.

**Parameters** **music** (`muspy.Music`) – Music object to convert.

**Returns** **multitrack** – Converted Multitrack object.

**Return type** `pypianoroll.Multitrack`

## 7.5 Datasets

MusPy provides an easy-to-use dataset management system. Each supported dataset comes with a class inherited from the base MusPy Dataset class. MusPy also provides interfaces to PyTorch and TensorFlow for creating input pipelines for machine learning. Here is an example of preparing training data in the piano-roll representation from the NES Music Database using MusPy.

```
import muspy

# Download and extract the dataset
nes = muspy.NESMusicDatabase("data/nas/", download_and_extract=True)

# Convert the dataset to MusPy Music objects
nes.convert()

# Iterate over the dataset
for music in nes:
    do_something(music)

# Convert to a PyTorch dataset
dataset = nes.to_pytorch_dataset(representation="pianoroll")
```

### 7.5.1 Iterating over a MusPy Dataset object

Here is an illustration of the two internal processing modes for iterating over a MusPy Dataset object.

## 7.5.2 Supported Datasets

Here is a list of the supported datasets.

Dataset	Format	Hours	Songs	Genre	Melody	Chords	Multitrack
Lakh MIDI Dataset	MIDI	>5000	174,533	misc	*	*	*
MAESTRO Dataset	MIDI	201.21	1,282	classical			
Wikifonia Lead Sheet Dataset	MusicXML	198.40	6,405	misc	O	O	
Essen Folk Song Dataset	ABC	56.62	9,034	folk	O	O	
NES Music Database	MIDI	46.11	5,278	game	O		O
MusicNet Dataset	MIDI	30.36	323	classical			*
Hymnal Tune Dataset	MIDI	18.74	1,756	hymn	O		
Hymnal Dataset	MIDI	17.50	1,723	hymn			
music21's Corpus	misc	16.86	613	misc	*		*
Nottingham Database	ABC	10.54	1,036	folk	O	O	
music21's JSBach Corpus	MusicXML	3.46	410	classical			O
JSBach Chorale Dataset	MIDI	3.21	382	classical			O
Haydn Op.20 Dataset	Humdrum	1.26	24	classical		O	

(Asterisk marks indicate partial support.)

## 7.5.3 Base Dataset Classes

Here are the two base classes for MusPy datasets.

**class** `muspy.Dataset`

Base class for MusPy datasets.

To build a custom dataset, it should inherit this class and override the methods `__getitem__` and `__len__` as well as the class attribute `_info`. `__getitem__` should return the *i*-th data sample as a `muspy.Music` object. `__len__` should return the size of the dataset. `_info` should be a `muspy.DatasetInfo` instance storing the dataset information.

**classmethod** `citation()`

Print the citation information.

**classmethod** `info()`

Return the dataset information.

**save** (*root*: `Union[str, pathlib.Path]`, *kind*: `Optional[str]` = 'json', *n\_jobs*: `int` = 1, *ignore\_exceptions*:

*bool* = True, *verbose*: *bool* = True, *\*\*kwargs*)

Save all the music objects to a directory.

### Parameters

- **root** (*str* or *Path*) – Root directory to save the data.
- **kind** ({'json', 'yaml'}, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`

## Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**split** (*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None*) → Dict[str, List[int]]  
Return the dataset as a PyTorch dataset.

### Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]  
Return the dataset as a PyTorch dataset.

### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or Dict of `:class:torch.utils.data.Dataset`

```
to_tensorflow_dataset (factory: Optional[Callable] = None, representation: Optional[str] = None, split_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None, **kwargs)  
    → Union[TFDataset, Dict[str, TFDataset]]
```

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable*, *optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str*, *optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str* or *Path*, *optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float* or *list of float*, *optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int*, *array\_like* or *RandomState*, *optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

#### Returns

- `class:tensorflow.data.Dataset` or `Dict` of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

```
class muspy.RemoteDataset (root: Union[str, pathlib.Path], download_and_extract: Optional[bool] = None, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, verbose: bool = True)
```

Base class for remote MusPy datasets.

This class extends `muspy.Dataset` to support remote datasets. To build a custom remote dataset, please refer to the documentation of `muspy.Dataset` for details. In addition, set the class attribute `_sources` to the URLs to the source files (see Notes).

#### root

Root directory of the dataset.

Type `str` or `Path`

#### Parameters

- **download\_and\_extract** (*bool*, *optional*) – Whether to download and extract the dataset. Defaults to False.
- **overwrite** (*bool*, *optional*) – Whether to overwrite existing downloaded files. Defaults to True.
- **cleanup** (*bool*, *optional*) – Whether to remove the source archive(s). Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Raises** `RuntimeError`: – If `download_and_extract` is False but file `{root}/.muspy.success` does not exist (see below).

---

**Important:** `muspy.Dataset.exists()` depends solely on a special file named `.muspy.success` in directory `{root}/_converted/`. This file serves as an indicator for the existence and integrity of the dataset. It will automatically be created if the dataset is successfully downloaded and extracted by `muspy.Dataset.download_and_extract()`. If the dataset is downloaded manually, make sure to create the `.muspy.success` file in directory `{root}/_converted/` to prevent errors.

---

## Notes

The class attribute `_sources` is a dictionary storing the following information of each source file.

- `filename (str)`: Name to save the file.
- `url (str)`: URL to the file.
- `archive (bool)`: Whether the file is an archive.
- `md5 (str, optional)`: Expected MD5 checksum of the file.

Here is an example.:

```
_sources = {
    "example": {
        "filename": "example.tar.gz",
        "url": "https://www.example.com/example.tar.gz",
        "archive": True,
        "md5": None,
    }
}
```

See also:

**`muspy.Dataset`** Base class for MusPy datasets.

**`classmethod citation()`**

Print the citation information.

**`download (overwrite: bool = True, verbose: bool = True)`** → `RemoteDatasetType`

Download the source datasets.

### Parameters

- **`overwrite (bool, optional)`** – Whether to overwrite existing files. Defaults to `True`.
- **`verbose (bool, optional)`** – Whether to be verbose. Defaults to `True`.

### Returns

**Return type** Object itself.

**`download_and_extract (overwrite: bool = True, cleanup: bool = False, verbose: bool = True)`** → `RemoteDatasetType`

Download source datasets and extract the downloaded archives.

### Parameters

- **`overwrite (bool, optional)`** – Whether to overwrite existing files. Defaults to `True`.

- **cleanup** (*bool*, *optional*) – Whether to remove the source archive. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**exists** () → bool

Return True if the dataset exists, otherwise False.

**extract** (*cleanup: bool = False*, *verbose: bool = True*) → RemoteDatasetType

Extract the downloaded archive(s).

**Parameters**

- **cleanup** (*bool*, *optional*) – Whether to remove the source archive after extraction. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**classmethod info** ()

Return the dataset information.

**save** (*root: Union[str, pathlib.Path]*, *kind: Optional[str] = 'json'*, *n\_jobs: int = 1*, *ignore\_exceptions: bool = True*, *verbose: bool = True*, *\*\*kwargs*)

Save all the music objects to a directory.

**Parameters**

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}*, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`

**Notes**

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**source\_exists** () → bool

Return True if all the sources exist, otherwise False.

**split** (*filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random\_state: Any = None*) → Dict[str, List[int]]

Return the dataset as a PyTorch dataset.

**Parameters**



- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or Dict of `:class:torch.utils.data.Dataset`

**to\_tensorflow\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and

test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random\_state** (*int*, *array\_like* or *RandomState*, *optional*) – Random state used to create the splits. If *int* or *array\_like*, the value is passed to `numpy.random.RandomState`, and the created *RandomState* object is used to create the splits. If *RandomState*, it will be used to create the splits.

#### Returns

- `class:tensorflow.data.Dataset` or `Dict` of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

## 7.5.4 Local Dataset Classes

Here are the classes for local datasets.

```
class muspy.FolderDataset (root: Union[str, pathlib.Path], convert: Optional[bool] = None, kind:
                           str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None,
                           use_converted: Optional[bool] = None)
```

Class for datasets storing files in a folder.

This class extends `muspy.Dataset` to support folder datasets. To build a custom folder dataset, please refer to the documentation of `muspy.Dataset` for details. In addition, set class attribute `_extension` to the extension to look for when building the dataset and set `read` to a callable that takes as inputs a filename of a source file and return the converted Music object.

#### root

Root directory of the dataset.

Type `str` or `Path`

#### Parameters

- **convert** (*bool*, *optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If *False*, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to *False*.
- **kind** (*{'json', 'yaml'}*, *optional*) – File format to save the data. Defaults to *'json'*.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to *True*.
- **use\_converted** (*bool*, *optional*) – Force to disable on-the-fly mode and use stored converted data

---

**Important:** `muspy.FolderDataset.converted_exists()` depends solely on a special file named `.muspy.success` in the folder `{root}/_converted/`, which serves as an indicator for the existence and integrity of the converted dataset. If the converted dataset is built by `muspy.FolderDataset.convert()`, the `.muspy.success` file will be created as well. If the converted dataset is created manually, make sure to create the `.muspy.success` file in the folder `{root}/_converted/` to prevent errors.

---

## Notes

Two modes are available for this dataset. When the on-the-fly mode is enabled, a data sample is converted to a music object on the fly when being indexed. When the on-the-fly mode is disabled, a data sample is loaded from the precomputed converted data.

See also:

**`muspy.Dataset`** Base class for MusPy datasets.

**`classmethod citation()`**

Print the citation information.

**`convert`** (*kind*: str = 'json', *n\_jobs*: int = 1, *ignore\_exceptions*: bool = True, *verbose*: bool = True) →

FolderDatasetType

Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

### Parameters

- **`kind`** ({'json', 'yaml'}, optional) – File format to save the data. Defaults to 'json'.
- **`n_jobs`** (int, optional) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **`ignore_exceptions`** (bool, optional) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **`verbose`** (bool, optional) – Whether to be verbose. Defaults to True.

### Returns

**Return type** Object itself.

**`converted_dir`**

Path to the root directory of the converted dataset.

**`converted_exists()`** → bool

Return True if the saved dataset exists, otherwise False.

**`exists()`** → bool

Return True if the dataset exists, otherwise False.

**`classmethod info()`**

Return the dataset information.

**`load`** (*filename*: Union[str, pathlib.Path]) → muspy.music.Music

Load a file into a Music object.

**`on_the_fly()`** → FolderDatasetType

Enable on-the-fly mode and convert the data on the fly.

### Returns

**Return type** Object itself.

**`read`** (*filename*: Any) → muspy.music.Music

Read a file into a Music object.

**save** (*root*: Union[str, pathlib.Path], *kind*: Optional[str] = 'json', *n\_jobs*: int = 1, *ignore\_exceptions*: bool = True, *verbose*: bool = True, *\*\*kwargs*)  
Save all the music objects to a directory.

#### Parameters

- **root** (*str* or *Path*) – Root directory to save the data.
- **kind** ({'json', 'yaml'}, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`

#### Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**split** (*filename*: Union[str, pathlib.Path, None] = None, *splits*: Optional[Sequence[float]] = None, *random\_state*: Any = None) → Dict[str, List[int]]  
Return the dataset as a PyTorch dataset.

#### Parameters

- **filename** (*str* or *Path*, *optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float* or *list of float*, *optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int*, *array\_like* or *RandomState*, *optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory*: Optional[Callable] = None, *representation*: Optional[str] = None, *split\_filename*: Union[str, pathlib.Path, None] = None, *splits*: Optional[Sequence[float]] = None, *random\_state*: Any = None, *\*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

#### Parameters

- **factory** (*Callable*, *optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str*, *optional*) – Target representation. See `muspy.to_representation()` for available representation.

- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or Dict of `:class:torch.utils.data.Dataset`

**to\_tensorflow\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*)  
 → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

#### Returns

- `class:tensorflow.data.Dataset` or Dict of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

**use\_converted** () → FolderDatasetType

Disable on-the-fly mode and use converted data.

#### Returns

**Return type** Object itself.

**class** `muspy.MusicDataset` (*root: Union[str, pathlib.Path], kind: str = 'json'*)

Class for datasets of MusPy JSON/YAML files.

**root**

Root directory of the dataset.

**Type** `str` or `Path`

**kind**

File format of the data. Defaults to 'json'.

**Type** {'json', 'yaml'}, optional

See also:

***muspy.Dataset*** Base class for MusPy datasets.

**classmethod citation()**

Print the citation information.

**classmethod info()**

Return the dataset information.

**save** (*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n\_jobs: int = 1, ignore\_exceptions: bool = True, verbose: bool = True, \*\*kwargs*)  
Save all the music objects to a directory.

**Parameters**

- **root** (*str or Path*) – Root directory to save the data.
- **kind** ({'json', 'yaml'}, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to *muspy.save\_json()* or *muspy.save\_yaml()*

**Notes**

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**split** (*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None*) → Dict[str, List[int]]  
Return the dataset as a PyTorch dataset.

**Parameters**

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or Dict of `:class:torch.utils.data.Dataset`

**to\_tensorflow\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

#### Returns

- `class:tensorflow.data.Dataset` or Dict of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

```
class muspy.ABCFolderDataset (root: Union[str, pathlib.Path], convert: Optional[bool] = None,  
                             kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Op-  
                             tional[bool] = None, use_converted: Optional[bool] = None)
```

Class for datasets storing ABC files in a folder.

See also:

[`muspy.FolderDataset`](#) Class for datasets storing files in a folder.

```
classmethod citation()
```

Print the citation information.

```
convert (kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, verbose: bool = True) →  
         FolderDatasetType
```

Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

#### Parameters

- **kind** (`{'json', 'yaml'}`, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

#### Returns

**Return type** Object itself.

```
converted_dir
```

Path to the root directory of the converted dataset.

```
converted_exists () → bool
```

Return True if the saved dataset exists, otherwise False.

```
exists () → bool
```

Return True if the dataset exists, otherwise False.

```
classmethod info()
```

Return the dataset information.

```
load (filename: Union[str, pathlib.Path]) → muspy.music.Music
```

Load a file into a Music object.

```
on_the_fly () → FolderDatasetType
```

Enable on-the-fly mode and convert the data on the fly.

#### Returns

**Return type** Object itself.

```
read (filename: Tuple[str, Tuple[int, int]]) → muspy.music.Music
```

Read a file into a Music object.



**save** (*root*: Union[str, pathlib.Path], *kind*: Optional[str] = 'json', *n\_jobs*: int = 1, *ignore\_exceptions*: bool = True, *verbose*: bool = True, *\*\*kwargs*)  
 Save all the music objects to a directory.

#### Parameters

- **root** (*str* or *Path*) – Root directory to save the data.
- **kind** ({'json', 'yaml'}, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`

#### Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**split** (*filename*: Union[str, pathlib.Path, None] = None, *splits*: Optional[Sequence[float]] = None, *random\_state*: Any = None) → Dict[str, List[int]]  
 Return the dataset as a PyTorch dataset.

#### Parameters

- **filename** (*str* or *Path*, *optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float* or *list of float*, *optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int*, *array\_like* or *RandomState*, *optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory*: Optional[Callable] = None, *representation*: Optional[str] = None, *split\_filename*: Union[str, pathlib.Path, None] = None, *splits*: Optional[Sequence[float]] = None, *random\_state*: Any = None, *\*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

#### Parameters

- **factory** (*Callable*, *optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str*, *optional*) – Target representation. See `muspy.to_representation()` for available representation.

- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or Dict of `:class:torch.utils.data.Dataset`

**to\_tensorflow\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*)  
→ Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

#### Returns

- `class:tensorflow.data.Dataset` or Dict of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

**use\_converted** () → FolderDatasetType

Disable on-the-fly mode and use converted data.

#### Returns

**Return type** Object itself.

## 7.5.5 Remote Dataset Classes

Here are the classes for remote datasets.

```
class muspy.RemoteFolderDataset(root: Union[str, pathlib.Path], download_and_extract: bool
                                = False, overwrite: Optional[bool] = None, cleanup: Op-
                                tional[bool] = None, convert: Optional[bool] = None, kind: str
                                = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] =
                                None, use_converted: Optional[bool] = None, verbose: bool =
                                True)
```

Base class for remote datasets storing files in a folder.

**root**

Root directory of the dataset.

Type `str` or `Path`

#### Parameters

- **download\_and\_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to `False`.
- **cleanup** (*bool, optional*) – Whether to remove the source archive(s). Defaults to `False`.
- **convert** (*bool, optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If `False`, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to `False`.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to `'json'`.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to `True`.
- **use\_converted** (*bool, optional*) – Force to disable on-the-fly mode and use stored converted data

See also:

**`muspy.FolderDataset`** Class for datasets storing files in a folder.

**`muspy.RemoteDataset`** Base class for remote MusPy datasets.

**`classmethod citation()`**

Print the citation information.

**`convert`** (*kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: bool = True, verbose: bool = True*) → `FolderDatasetType`

Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

#### Parameters

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to `'json'`.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.

- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**converted\_dir**

Path to the root directory of the converted dataset.

**converted\_exists** () → bool

Return True if the saved dataset exists, otherwise False.

**download** (*overwrite: bool = True*, *verbose: bool = True*) → RemoteDatasetType

Download the source datasets.

**Parameters**

- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**download\_and\_extract** (*overwrite: bool = True*, *cleanup: bool = False*, *verbose: bool = True*) → RemoteDatasetType

Download source datasets and extract the downloaded archives.

**Parameters**

- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files. Defaults to True.
- **cleanup** (*bool*, *optional*) – Whether to remove the source archive. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**exists** () → bool

Return True if the dataset exists, otherwise False.

**extract** (*cleanup: bool = False*, *verbose: bool = True*) → RemoteDatasetType

Extract the downloaded archive(s).

**Parameters**

- **cleanup** (*bool*, *optional*) – Whether to remove the source archive after extraction. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**classmethod info** ()

Return the dataset information.

**load** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music

Load a file into a Music object.

**on\_the\_fly** () → FolderDatasetType

Enable on-the-fly mode and convert the data on the fly.

### Returns

**Return type** Object itself.

**read** (*filename: str*) → muspy.music.Music

Read a file into a Music object.

**save** (*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n\_jobs: int = 1, ignore\_exceptions:*

*bool = True, verbose: bool = True, \*\*kwargs*)

Save all the music objects to a directory.

### Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`

### Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**source\_exists** () → bool

Return True if all the sources exist, otherwise False.

**split** (*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None,*

*random\_state: Any = None*) → Dict[str, List[int]]

Return the dataset as a PyTorch dataset.

### Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

```
to_pytorch_dataset (factory: Optional[Callable] = None, representation: Optional[str] = None, split_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None, **kwargs) → Union[TorchDataset, Dict[str, TorchDataset]]
```

Return the dataset as a PyTorch dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or Dict of `:class:torch.utils.data.Dataset`

```
to_tensorflow_dataset (factory: Optional[Callable] = None, representation: Optional[str] = None, split_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None, **kwargs) → Union[TFDataset, Dict[str, TFDataset]]
```

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

#### Returns

- `class:tensorflow.data.Dataset` or Dict of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

**use\_converted**() → FolderDatasetType  
 Disable on-the-fly mode and use converted data.

#### Returns

**Return type** Object itself.

**class** muspy.RemoteMusicDataset(*root: Union[str, pathlib.Path], download\_and\_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, kind: str = 'json', verbose: bool = True*)

Base class for remote datasets of MusPy JSON/YAML files.

**root**  
 Root directory of the dataset.

**Type** str or Path

**kind**  
 File format of the data. Defaults to 'json'.

**Type** {'json', 'yaml'}, optional

#### Parameters

- **download\_and\_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the source archive(s). Defaults to False.

See also:

***muspy.MusicDataset*** Class for datasets of MusPy JSON/YAML files.

***muspy.RemoteDataset*** Base class for remote MusPy datasets.

**classmethod** **citation**()  
 Print the citation information.

**download** (*overwrite: bool = True, verbose: bool = True*) → RemoteDatasetType  
 Download the source datasets.

#### Parameters

- **overwrite** (*bool, optional*) – Whether to overwrite existing files. Defaults to True.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.

#### Returns

**Return type** Object itself.

**download\_and\_extract** (*overwrite: bool = True, cleanup: bool = False, verbose: bool = True*) → RemoteDatasetType  
 Download source datasets and extract the downloaded archives.

#### Parameters

- **overwrite** (*bool, optional*) – Whether to overwrite existing files. Defaults to True.
- **cleanup** (*bool, optional*) – Whether to remove the source archive. Defaults to False.

- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**exists** () → bool

Return True if the dataset exists, otherwise False.

**extract** (*cleanup*: bool = False, *verbose*: bool = True) → RemoteDatasetType

Extract the downloaded archive(s).

**Parameters**

- **cleanup** (*bool*, *optional*) – Whether to remove the source archive after extraction. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**classmethod info** ()

Return the dataset information.

**save** (*root*: Union[str, pathlib.Path], *kind*: Optional[str] = 'json', *n\_jobs*: int = 1, *ignore\_exceptions*: bool = True, *verbose*: bool = True, *\*\*kwargs*)

Save all the music objects to a directory.

**Parameters**

- **root** (*str* or *Path*) – Root directory to save the data.
- **kind** ({'json', 'yaml'}, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`

**Notes**

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**source\_exists** () → bool

Return True if all the sources exist, otherwise False.

**split** (*filename*: Union[str, pathlib.Path, None] = None, *splits*: Optional[Sequence[float]] = None, *random\_state*: Any = None) → Dict[str, List[int]]

Return the dataset as a PyTorch dataset.

**Parameters**

- **filename** (*str* or *Path*, *optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.



- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or Dict of `:class:torch.utils.data.Dataset`

**to\_tensorflow\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random\_state** (*int*, *array\_like* or *RandomState*, *optional*) – Random state used to create the splits. If *int* or *array\_like*, the value is passed to `numpy.random.RandomState`, and the created *RandomState* object is used to create the splits. If *RandomState*, it will be used to create the splits.

#### Returns

- `class:tensorflow.data.Dataset` or Dict of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

```
class muspy.RemoteABCFolderDataset(root: Union[str, pathlib.Path], download_and_extract:  
                                bool = False, overwrite: Optional[bool] = None, cleanup:  
                                Optional[bool] = None, convert: Optional[bool] = None,  
                                kind: str = 'json', n_jobs: int = 1, ignore_exceptions:  
                                Optional[bool] = None, use_converted: Optional[bool] =  
                                None, verbose: bool = True)
```

Base class for remote datasets storing ABC files in a folder.

See also:

**`muspy.ABCFolderDataset`** Class for datasets storing ABC files in a folder.

**`muspy.RemoteDataset`** Base class for remote MusPy datasets.

**`classmethod citation()`**

Print the citation information.

**`convert`** (*kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: bool = True, verbose: bool = True*) →

FolderDatasetType

Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

#### Parameters

- **`kind`** (*{'json', 'yaml'}*, *optional*) – File format to save the data. Defaults to 'json'.
- **`n_jobs`** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **`ignore_exceptions`** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **`verbose`** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

#### Returns

**Return type** Object itself.

**`converted_dir`**

Path to the root directory of the converted dataset.

**`converted_exists()`** → bool

Return True if the saved dataset exists, otherwise False.

**`download`** (*overwrite: bool = True, verbose: bool = True*) → RemoteDatasetType

Download the source datasets.

#### Parameters

- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns****Return type** Object itself.

**download\_and\_extract** (*overwrite: bool = True, cleanup: bool = False, verbose: bool = True*) → RemoteDatasetType  
Download source datasets and extract the downloaded archives.

**Parameters**

- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files. Defaults to True.
- **cleanup** (*bool*, *optional*) – Whether to remove the source archive. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns****Return type** Object itself.

**exists** () → bool  
Return True if the dataset exists, otherwise False.

**extract** (*cleanup: bool = False, verbose: bool = True*) → RemoteDatasetType  
Extract the downloaded archive(s).

**Parameters**

- **cleanup** (*bool*, *optional*) – Whether to remove the source archive after extraction. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns****Return type** Object itself.

**classmethod info** ()  
Return the dataset information.

**load** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Load a file into a Music object.

**on\_the\_fly** () → FolderDatasetType  
Enable on-the-fly mode and convert the data on the fly.

**Returns****Return type** Object itself.

**read** (*filename: Tuple[str, Tuple[int, int]]*) → muspy.music.Music  
Read a file into a Music object.

**save** (*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n\_jobs: int = 1, ignore\_exceptions: bool = True, verbose: bool = True, \*\*kwargs*)  
Save all the music objects to a directory.

**Parameters**

- **root** (*str or Path*) – Root directory to save the data.

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`

## Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**source\_exists()** → bool

Return True if all the sources exist, otherwise False.

**split** (*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None*) → Dict[str, List[int]]

Return the dataset as a PyTorch dataset.

### Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and

test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If *int* or *array\_like*, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or `Dict of :class:torch.utils.data.Dataset`

**to\_tensorflow\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*)  
→ `Union[TFDataset, Dict[str, TFDataset]]`

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If *None* or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If *None*, return the full dataset as a whole. If *float*, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If *int* or *array\_like*, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

#### Returns

- `class:tensorflow.data.Dataset` or `Dict of`
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

**use\_converted** () → `FolderDatasetType`

Disable on-the-fly mode and use converted data.

#### Returns

**Return type** Object itself.

## 7.6 Representations

MusPy supports several common representations for symbolic music. Here is a comparison of them.

Representation	Shape	Values	Default configurations
Pitch-based	$T \times 1$	$\{0, 1, \dots, 129\}$	128 note-ons, 1 hold, 1 rest (support only monophonic music)
Piano-roll	$T \times 128$	$\{0, 1\}$ or $N$	$\{0,1\}$ for binary piano rolls; $N$ for piano rolls with velocities
Event-based	$M \times 1$	$\{0, 1, \dots, 387\}$	128 note-ons, 128 note-offs, 100 tick shifts, 32 velocities
Note-based	$N \times 4$	$N$	List of $(time, pitch, duration, velocity)$ tuples

Note that  $T$ ,  $M$ , and  $N$  denote the numbers of time steps, events and notes, respectively.

MusPy's representation module supports two types of two APIs—Functional API and Processor API. Take the pitch-based representation for example.

- The Functional API provide two functions: - `muspy.to_pitch_representation()`: Convert a Music object into pitch-based representation - `muspy.from_pitch_representation()`: Return a Music object converted from pitch-based representation
- The Processor API provides the class `muspy.PitchRepresentationProcessor`, which provides two methods: - `muspy.PitchRepresentationProcessor.encode()`: Convert a Music object into pitch-based representation - `muspy.PitchRepresentationProcessor.decode()`: Return a Music object converted from pitch-based representation

## 7.6.1 Pitch-based Representation

`muspy.to_pitch_representation(music: Music, use_hold_state: bool = False) → numpy.ndarray`  
Encode a Music object into pitch-based representation.

The pitch-based representation represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is  $T \times 1$ , where  $T$  is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

### Parameters

- **music** (`muspy.Music`) – Music object to encode.
- **use\_hold\_state** (`bool`) – Whether to use a special state for holds. Defaults to False.

**Returns** Encoded array in pitch-based representation.

**Return type** ndarray, dtype=uint8, shape=(?, 1)

`muspy.from_pitch_representation(array: numpy.ndarray, resolution: int = 24, program: int = 0, is_drum: bool = False, use_hold_state: bool = False, default_velocity: int = 64) → muspy.music.Music`

Decode pitch-based representation into a Music object.

### Parameters

- **array** (`ndarray`) – Array in pitch-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (`int`) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.
- **program** (`int`, *optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (`bool`, *optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_hold\_state** (`bool`) – Whether to use a special state for holds. Defaults to False.

- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** Decoded Music object.

**Return type** *muspy.Music*

## References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

**class** *muspy.PitchRepresentationProcessor* (*use\_hold\_state: bool = False, default\_velocity: int = 64*)

Pitch-based representation processor.

The pitch-based representation represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is T x 1, where T is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

**use\_hold\_state**

Whether to use a special state for holds. Defaults to False.

**Type** *bool*

**default\_velocity**

Default velocity value to use when decoding. Defaults to 64.

**Type** *int*

**decode** (*array: numpy.ndarray*) → *muspy.music.Music*

Decode pitch-based representation into a Music object.

**Parameters** *array* (*ndarray*) – Array in pitch-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** *muspy.Music* object

**See also:**

*muspy.from\_pitch\_representation()* Return a Music object converted from pitch-based representation.

**encode** (*music: muspy.music.Music*) → *numpy.ndarray*

Encode a Music object into pitch-based representation.

**Parameters** *music* (*muspy.Music* object) – Music object to encode.

**Returns** Encoded array in pitch-based representation.

**Return type** *ndarray* (*np.uint8*)

**See also:**

*muspy.to\_pitch\_representation()* Convert a Music object into pitch-based representation.

## 7.6.2 Piano-roll Representation

`muspy.to_pianoroll_representation` (*music*: *Music*, *encode\_velocity*: *bool* = *True*) → *numpy.ndarray*

Encode notes into piano-roll representation.

### Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **encode\_velocity** (*bool*) – Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

**Returns** Encoded array in piano-roll representation.

**Return type** *ndarray*, *dtype*=*uint8* or *bool*, *shape*=(?, 128)

`muspy.from_pianoroll_representation` (*array*: *numpy.ndarray*, *resolution*: *int* = 24, *program*: *int* = 0, *is\_drum*: *bool* = *False*, *encode\_velocity*: *bool* = *True*, *default\_velocity*: *int* = 64) → *muspy.music.Music*

Decode pitch-based representation into a Music object.

### Parameters

- **array** (*ndarray*) – Array in piano-roll representation to decode. Will be casted to integer if not of integer type. If *encode\_velocity* is True, will be casted to boolean if not of boolean type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.
- **program** (*int*, *optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool*, *optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode velocities. Defaults to True.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** Decoded Music object.

**Return type** *muspy.Music*

## References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

**class** `muspy.PianoRollRepresentationProcessor` (*encode\_velocity*: *bool* = *True*, *default\_velocity*: *int* = 64)

Piano-roll representation processor.

The piano-roll representation represents music as a time-pitch matrix, where the columns are the time steps and the rows are the pitches. The values indicate the presence of pitches at different time steps. The output shape is T x 128, where T is the number of time steps.

### **encode\_velocity**

Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

**Type** *bool*



**default\_velocity**

Default velocity value to use when decoding if *encode\_velocity* is False. Defaults to 64.

Type `int`

**decode** (*array: numpy.ndarray*) → *muspy.music.Music*

Decode piano-roll representation into a Music object.

**Parameters** *array* (*ndarray*) – Array in piano-roll representation to decode. Cast to integer if not of integer type. If *encode\_velocity* is True, casted to boolean if not of boolean type.

**Returns** Decoded Music object.

**Return type** *muspy.Music* object

See also:

*muspy.from\_pianoroll\_representation()* Return a Music object converted from piano-roll representation.

**encode** (*music: muspy.music.Music*) → *numpy.ndarray*

Encode a Music object into piano-roll representation.

**Parameters** *music* (*muspy.Music* object) – Music object to encode.

**Returns** Encoded array in piano-roll representation.

**Return type** *ndarray* (*np.uint8*)

See also:

*muspy.to\_pianoroll\_representation()* Convert a Music object into piano-roll representation.

### 7.6.3 Event-based Representation

*muspy.to\_event\_representation* (*music: Music, use\_single\_note\_off\_event: bool = False, use\_end\_of\_sequence\_event: bool = False, encode\_velocity: bool = False, force\_velocity\_event: bool = True, max\_time\_shift: int = 100, velocity\_bins: int = 32*) → *numpy.ndarray*

Encode a Music object into event-based representation.

The event-based representation represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is *M* x 1, where *M* is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-on events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

**Parameters**

- **music** (*muspy.Music*) – Music object to encode.
- **use\_single\_note\_off\_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use\_end\_of\_sequence\_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode velocities.

- **force\_velocity\_event** (*bool*) – Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.
- **max\_time\_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max\_time\_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity\_bins** (*int*) – Number of velocity bins to use. Defaults to 32.

**Returns** Encoded array in event-based representation.

**Return type** ndarray, dtype=uint16, shape=(?, 1)

```
muspy.from_event_representation(array: numpy.ndarray, resolution: int = 24, program: int = 0, is_drum: bool = False, use_single_note_off_event: bool = False, use_end_of_sequence_event: bool = False, max_time_shift: int = 100, velocity_bins: int = 32, default_velocity: int = 64, duplicate_note_mode: str = 'fifo') → muspy.music.Music
```

Decode event-based representation into a Music object.

#### Parameters

- **array** (*ndarray*) – Array in event-based representation to decode. Cast to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.
- **program** (*int*, *optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool*, *optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_single\_note\_off\_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, a note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use\_end\_of\_sequence\_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **max\_time\_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max\_time\_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity\_bins** (*int*) – Number of velocity bins to use. Defaults to 32.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.
- **duplicate\_note\_mode** (*{'fifo', 'lifo', 'close\_all'}*) – Policy for dealing with duplicate notes. When a note off event is presetned while there are multiple corresponsing note on events that have not yet been closed, we need a policy to decide which note on messages to close. This is only effective when *use\_single\_note\_off\_event* is False. Defaults to 'fifo'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out): close the latest note on
  - 'close\_all': close all note on messages

**Returns** Decoded Music object.

**Return type** `muspy.Music`

## References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

```
class muspy.EventRepresentationProcessor (use_single_note_off_event: bool = False,
                                         use_end_of_sequence_event: bool =
                                         False, encode_velocity: bool = False,
                                         force_velocity_event: bool = True,
                                         max_time_shift: int = 100, velocity_bins:
                                         int = 32, default_velocity: int = 64)
```

Event-based representation processor.

The event-based representation represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is  $M \times 1$ , where  $M$  is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-on events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

### **use\_single\_note\_off\_event**

Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.

**Type** `bool`

### **use\_end\_of\_sequence\_event**

Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.

**Type** `bool`

### **encode\_velocity**

Whether to encode velocities.

**Type** `bool`

### **force\_velocity\_event**

Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.

**Type** `bool`

### **max\_time\_shift**

Maximum time shift (in ticks) to be encoded as a separate event. Time shifts larger than `max_time_shift` will be decomposed into two or more time-shift events. Defaults to 100.

**Type** `int`

### **velocity\_bins**

Number of velocity bins to use. Defaults to 32.

**Type** `int`

### **default\_velocity**

Default velocity value to use when decoding. Defaults to 64.

**Type** `int`

**decode** (`array: numpy.ndarray`)  $\rightarrow$  `muspy.music.Music`

Decode event-based representation into a Music object.

**Parameters** **array** (*ndarray*) – Array in event-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** *muspy.Music* object

**See also:**

*muspy.from\_event\_representation()* Return a Music object converted from event-based representation.

**encode** (*music: muspy.music.Music*) → *numpy.ndarray*  
Encode a Music object into event-based representation.

**Parameters** **music** (*muspy.Music* object) – Music object to encode.

**Returns** Encoded array in event-based representation.

**Return type** *ndarray* (*np.uint16*)

**See also:**

*muspy.to\_event\_representation()* Convert a Music object into event-based representation.

## 7.6.4 Note-based Representation

**muspy.to\_note\_representation** (*music: Music, use\_start\_end: bool = False, encode\_velocity: bool = True, dtype: Union[numpy.dtype, type, str] = <class 'int'>*) → *numpy.ndarray*

Encode a Music object into note-based representation.

The note-based representation represents music as a sequence of (time, pitch, duration, velocity) tuples. For example, a note *Note*(time=0, duration=4, pitch=60, velocity=64) will be encoded as a tuple (0, 60, 4, 64). The output shape is *N \* D*, where *N* is the number of notes and *D* is 4 when *encode\_velocity* is *True*, otherwise *D* is 3. The values of the second dimension represent time, pitch, duration and velocity (discarded when *encode\_velocity* is *False*).

### Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use\_start\_end** (*bool*) – Whether to use ‘start’ and ‘end’ to encode the timing rather than ‘time’ and ‘duration’. Defaults to *False*.
- **encode\_velocity** (*bool*) – Whether to encode note velocities. Defaults to *True*.
- **dtype** (*dtype, type or str*) – Data type of the return array. Defaults to *int*.

**Returns** Encoded array in note-based representation.

**Return type** *ndarray*, shape=(?, 3 or 4)

**muspy.from\_note\_representation** (*array: numpy.ndarray, resolution: int = 24, program: int = 0, is\_drum: bool = False, use\_start\_end: bool = False, encode\_velocity: bool = True, default\_velocity: int = 64*) → *muspy.music.Music*

Decode note-based representation into a Music object.

### Parameters

- **array** (*ndarray*) – Array in note-based representation to decode. Will be casted to integer if not of integer type.

- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.
- **program** (*int*, *optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool*, *optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_start\_end** (*bool*) – Whether to use ‘start’ and ‘end’ to encode the timing rather than ‘time’ and ‘duration’. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **default\_velocity** (*int*) – Default velocity value to use when decoding if *encode\_velocity* is False. Defaults to 64.

**Returns** Decoded Music object.

**Return type** *muspy.Music*

## References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

```
class muspy.NoteRepresentationProcessor(use_start_end: bool = False, encode_velocity:  
                                         bool = True, dtype: Union[numpy.dtype, type, str]  
                                         = <class 'int'>, default_velocity: int = 64)
```

Note-based representation processor.

The note-based representation represents music as a sequence of (pitch, time, duration, velocity) tuples. For example, a note `Note(time=0, duration=4, pitch=60, velocity=64)` will be encoded as a tuple `(0, 4, 60, 64)`. The output shape is `L * D`, where `L` is the number of notes and `D` is 4 when *encode\_velocity* is True, otherwise `D` is 3. The values of the second dimension represent pitch, time, duration and velocity (discarded when *encode\_velocity* is False).

**use\_start\_end**

Whether to use ‘start’ and ‘end’ to encode the timing rather than ‘time’ and ‘duration’. Defaults to False.

**Type** *bool*

**encode\_velocity**

Whether to encode note velocities. Defaults to True.

**Type** *bool*

**dtype**

Data type of the return array. Defaults to `int`.

**Type** *dtype, type or str*

**default\_velocity**

Default velocity value to use when decoding if *encode\_velocity* is False. Defaults to 64.

**Type** *int*

**decode** (*array: numpy.ndarray*) → *muspy.music.Music*

Decode note-based representation into a Music object.

**Parameters** **array** (*ndarray*) – Array in note-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** `muspy.Music` object

**See also:**

`muspy.from_note_representation()` Return a Music object converted from note-based representation.

**encode** (*music*: `muspy.music.Music`) → `numpy.ndarray`  
Encode a Music object into note-based representation.

**Parameters** **music** (`muspy.Music` object) – Music object to encode.

**Returns** Encoded array in note-based representation.

**Return type** `ndarray (np.uint8)`

**See also:**

`muspy.to_note_representation()` Convert a Music object into note-based representation.

## 7.7 Synthesis

`muspy.write_audio` (*path*: `Union[str, pathlib.Path]`, *music*: `Music`, *soundfont\_path*: `Union[str, pathlib.Path, None]` = `None`, *rate*: `int` = `44100`, *audio\_format*: `Optional[str]` = `None`)  
Write a Music object to an audio file.

Supported formats include WAV, AIFF, FLAC and OGA.

**Parameters**

- **path** (`str` or `Path`) – Path to write the audio file.
- **music** (`muspy.Music`) – Music object to write.
- **soundfont\_path** (`str` or `Path`, *optional*) – Path to the soundfont file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (`int`) – Sample rate (in samples per sec). Defaults to 44100.
- **audio\_format** (`str`, `{'wav', 'aiff', 'flac', 'oga'}`, *optional*) – File format to write. If `None`, infer it from the extension.

`muspy.synthesize` (*music*: `Music`, *soundfont\_path*: `Union[str, pathlib.Path, None]` = `None`, *rate*: `int` = `44100`) → `numpy.ndarray`  
Synthesize a Music object to raw audio.

**Parameters**

- **music** (`muspy.Music`) – Music object to write.
- **soundfont\_path** (`str` or `Path`, *optional*) – Path to the soundfont file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (`int`) – Sample rate (in samples per sec). Defaults to 44100.

**Returns** Synthesized waveform.

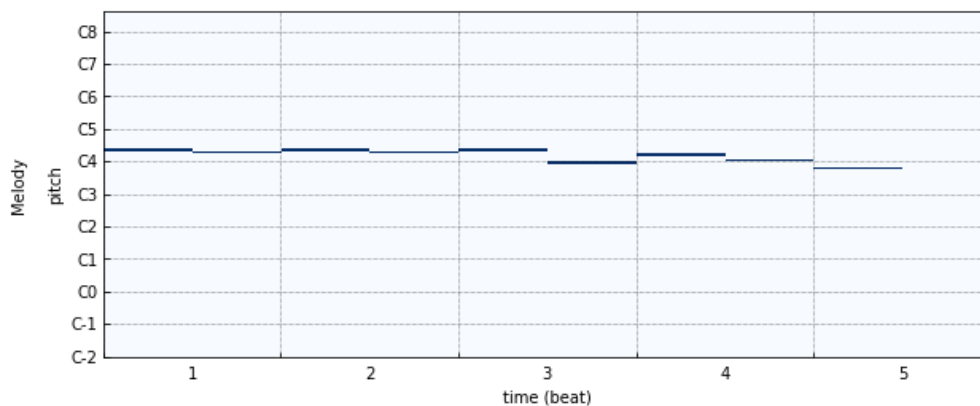
**Return type** `ndarray, dtype=int16, shape=(?, 2)`

## 7.8 Visualization

MusPy supports two visualization tools. Both use Matplotlib as the backend for flexibility.

### 7.8.1 Piano-roll Visualization

The piano-roll visualization is made possible with the [Pypianoroll](#) library.



```
muspy.show_pianoroll(music: Music, **kwargs)
```

Show pianoroll visualization.

### 7.8.2 Score Visualization

The score visualization is made possible with the [Bravura](#) font.



```
muspy.show_score(music: Music, figsize: Optional[Tuple[float, float]] = None, clef: str = 'treble',
                 clef_octave: Optional[int] = 0, note_spacing: Optional[int] = None, font_path:
                 Union[str, pathlib.Path, None] = None, font_scale: Optional[float] = None) →
                 muspy.visualization.score.ScorePlotter
```

Show score visualization.

#### Parameters

- **music** (*muspy.Music*) – Music object to show.
- **figsize** ((*float*, *float*), *optional*) – Width and height in inches. Defaults to Matplotlib configuration.
- **clef** (*str*, {'treble', 'alto', 'bass'}) – Clef type. Defaults to a treble clef.
- **clef\_octave** (*int*) – Clef octave. Defaults to zero.
- **note\_spacing** (*int*, *optional*) – Spacing of notes. Defaults to 4.

- **font\_path** (*str* or *Path*, *optional*) – Path to the music font. Defaults to the path to the built-in Bravura font.
- **font\_scale** (*float*, *optional*) – Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

**Returns** A ScorePlotter object that handles the score.

**Return type** `muspy.ScorePlotter`

`muspy.ScorePlotter` (*fig*: `matplotlib.figure.Figure`, *ax*: `matplotlib.axes._axes.Axes`, *resolution*: *int*, *note\_spacing*: *Optional[int] = None*, *font\_path*: *Union[str, pathlib.Path, None] = None*, *font\_scale*: *Optional[float] = None*)

A plotter that handles the score visualization.

`muspy.fig`

Figure object to plot the score on.

**Type** `matplotlib.figure.Figure`

`muspy.axes`

Axes object to plot the score on.

**Type** `matplotlib.axes.Axes`

`muspy.resolution`

Time steps per quarter note.

**Type** `int`

`muspy.note_spacing`

Spacing of notes. Defaults to 4.

**Type** `int`, optional

`muspy.font_path`

Path to the music font. Defaults to the path to the downloaded Bravura font.

**Type** `str` or `Path`, optional

`muspy.font_scale`

Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

**Type** `float`, optional

## 7.9 Metrics

MusPy provides several objective metrics proposed in the literature, summarized as follows.

- **Pitch-related metrics:** `pitch_range`, `n_pitches_used`, `n_pitch_classes_used`, `polyphony`, `polyphony_rate`, `pitch_in_scale_rate`, `scale_consistency`, `pitch_entropy` and `pitch_class_entropy`.
- **Rhythm-related metrics:** `empty_beat_rate`, `drum_in_pattern_rate`, `drum_pattern_consistency` and `groove_consistency`.
- **Other metrics:** `empty_measure_rate`.

These objective metrics could be used to evaluate a music generation system by comparing the statistical difference between the training data and the generated samples.



## 7.9.1 Pitch-related metrics

`muspy.pitch_range(music: muspy.music.Music) → int`

Return the pitch range.

Drum tracks are ignored. Return zero if no note is found.

**Parameters** `music` (*`muspy.Music`*) – Music object to evaluate.

**Returns** Pitch range.

**Return type** `int`

`muspy.n_pitches_used(music: muspy.music.Music) → int`

Return the number of unique pitches used.

Drum tracks are ignored.

**Parameters** `music` (*`muspy.Music`*) – Music object to evaluate.

**Returns** Number of unique pitch used.

**Return type** `int`

See also:

`muspy.n_pitch_class_used()` Compute the number of unique pitch classes used.

`muspy.n_pitch_classes_used(music: muspy.music.Music) → int`

Return the number of unique pitch classes used.

Drum tracks are ignored.

**Parameters** `music` (*`muspy.Music`*) – Music object to evaluate.

**Returns** Number of unique pitch classes used.

**Return type** `int`

See also:

`muspy.n_pitches_used()` Compute the number of unique pitches used.

`muspy.polyphony(music: muspy.music.Music) → float`

Return the average number of pitches being played concurrently.

The polyphony is defined as the average number of pitches being played at the same time, evaluated only at time steps where at least one pitch is on. Drum tracks are ignored. Return NaN if no note is found.

$$\text{polyphony} = \frac{\#(\text{pitches\_when\_at\_least\_one\_pitch\_is\_on})}{\#(\text{time\_steps\_where\_at\_least\_one\_pitch\_is\_on})}$$

**Parameters** `music` (*`muspy.Music`*) – Music object to evaluate.

**Returns** Polyphony.

**Return type** `float`

See also:

`muspy.polyphony_rate()` Compute the ratio of time steps where multiple pitches are on.

`muspy.polyphony_rate(music: muspy.music.Music, threshold: int = 2) → float`

Return the ratio of time steps where multiple pitches are on.

The polyphony rate is defined as the ratio of the number of time steps where multiple pitches are on to the total number of time steps. Drum tracks are ignored. Return NaN if song length is zero. This metric is used in [1], where it is called *polyphonicity*.

$$\text{polyphony\_rate} = \frac{\#(\text{time\_steps\_where\_multiple\_pitches\_are\_on})}{\#(\text{time\_steps})}$$

#### Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **threshold** (*int*) – Threshold of number of pitches to count into the numerator.

**Returns** Polyphony rate.

**Return type** `float`

See also:

***muspy.polyphony()*** Compute the average number of pitches being played at the same time.

#### References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

***muspy.pitch\_in\_scale\_rate*** (*music: muspy.music.Music, root: int, mode: str*) → `float`

Return the ratio of pitches in a certain musical scale.

The pitch-in-scale rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$\text{pitch\_in\_scale\_rate} = \frac{\#(\text{notes\_in\_scale})}{\#(\text{notes})}$$

#### Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **root** (*int*) – Root of the scale.
- **mode** (*str*, {*'major'*, *'minor'*}) – Mode of the scale.

**Returns** Pitch-in-scale rate.

**Return type** `float`

See also:

***muspy.scale\_consistency()*** Compute the largest pitch-in-class rate.

#### References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.scale_consistency(music: muspy.music.Music) → float`

Return the largest pitch-in-scale rate.

The scale consistency is defined as the largest pitch-in-scale rate over all major and minor scales. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$scale\_consistency = \max_{root, mode} pitch\_in\_scale\_rate(root, mode)$$

**Parameters** `music` (`muspy.Music`) – Music object to evaluate.

**Returns** Scale consistency.

**Return type** `float`

**See also:**

`muspy.pitch_in_scale_rate()` Compute the ratio of pitches in a certain musical scale.

## References

1. Olof Mogren, “C-RNN-GAN: Continuous recurrent neural networks with adversarial training,” in NeuIPS Workshop on Constructive Machine Learning, 2016.

`muspy.pitch_entropy(music: muspy.music.Music) → float`

Return the entropy of the normalized note pitch histogram.

The pitch entropy is defined as the Shannon entropy of the normalized note pitch histogram. Drum tracks are ignored. Return NaN if no note is found.

$$pitch\_entropy = - \sum_{i=0}^{127} P(pitch = i) \log_2 P(pitch = i)$$

**Parameters** `music` (`muspy.Music`) – Music object to evaluate.

**Returns** Pitch entropy.

**Return type** `float`

**See also:**

`muspy.pitch_class_entropy()` Compute the entropy of the normalized pitch class histogram.

`muspy.pitch_class_entropy(music: muspy.music.Music) → float`

Return the entropy of the normalized note pitch class histogram.

The pitch class entropy is defined as the Shannon entropy of the normalized note pitch class histogram. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$pitch\_class\_entropy = - \sum_{i=0}^{11} P(pitch\_class = i) \times \log_2 P(pitch\_class = i)$$

**Parameters** `music` (`muspy.Music`) – Music object to evaluate.

**Returns** Pitch class entropy.

**Return type** `float`

**See also:**

`muspy.pitch_entropy()` Compute the entropy of the normalized pitch histogram.

## References

1. Shih-Lun Wu and Yi-Hsuan Yang, “The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures”, in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

### 7.9.2 Rhythm-related metrics

`muspy.empty_beat_rate` (*music*: `muspy.music.Music`) → float

Return the ratio of empty beats.

The empty-beat rate is defined as the ratio of the number of empty beats (where no note is played) to the total number of beats. Return NaN if song length is zero. This metric is also implemented in Pypianoroll [1].

$$\text{empty\_beat\_rate} = \frac{\#(\text{empty\_beats})}{\#(\text{beats})}$$

**Parameters** `music` (`muspy.Music`) – Music object to evaluate.

**Returns** Empty-beat rate.

**Return type** float

See also:

`muspy.empty_measure_rate()` Compute the ratio of empty measures.

## References

1. Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang, “Pypianoroll: Open Source Python Package for Handling Multitrack Pianorolls,” in Late-Breaking Demos of the 18th International Society for Music Information Retrieval Conference (ISMIR), 2018.

`muspy.drum_in_pattern_rate` (*music*: `muspy.music.Music`, *meter*: `str`) → float

Return the ratio of drum notes in a certain drum pattern.

The drum-in-pattern rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Only drum tracks are considered. Return NaN if no drum note is found. This metric is used in [1].

$$\text{drum\_in\_pattern\_rate} = \frac{\#(\text{drum\_notes\_in\_pattern})}{\#(\text{drum\_notes})}$$

**Parameters**

- `music` (`muspy.Music`) – Music object to evaluate.
- `meter` (`str`, {'duple', 'triple'}) – Meter of the drum pattern.

**Returns** Drum-in-pattern rate.

**Return type** float

See also:

`muspy.drum_pattern_consistency()` Compute the largest drum-in-pattern rate.

## References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.drum_pattern_consistency` (*music*: *muspy.music.Music*) → float

Return the largest drum-in-pattern rate.

The drum pattern consistency is defined as the largest drum-in-pattern rate over duple and triple meters. Only drum tracks are considered. Return NaN if no drum note is found.

$$\text{drum\_pattern\_consistency} = \max_{\text{meter}} \text{drum\_in\_pattern\_rate}(\text{meter})$$

**Parameters** `music` (*muspy.Music*) – Music object to evaluate.

**Returns** Drum pattern consistency.

**Return type** float

**See also:**

`muspy.drum_in_pattern_rate()` Compute the ratio of drum notes in a certain drum pattern.

`muspy.groove_consistency` (*music*: *muspy.music.Music*, *measure\_resolution*: int) → float

Return the groove consistency.

The groove consistency is defined as the mean hamming distance of the neighboring measures.

$$\text{groove\_consistency} = 1 - \frac{1}{T-1} \sum_{i=1}^{T-1} d(G_i, G_{i+1})$$

Here,  $T$  is the number of measures,  $G_i$  is the binary onset vector of the  $i$ -th measure (a one at position that has an onset, otherwise a zero), and  $d(G, G')$  is the hamming distance between two vectors  $G$  and  $G'$ . Note that this metric only works for songs with a constant time signature. Return NaN if the number of measures is less than two. This metric is used in [1].

**Parameters**

- `music` (*muspy.Music*) – Music object to evaluate.
- `measure_resolution` (int) – Time steps per measure.

**Returns** Groove consistency.

**Return type** float

## References

1. Shih-Lun Wu and Yi-Hsuan Yang, “The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures”, in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

### 7.9.3 Other metrics

`muspy.empty_measure_rate` (*music*: *muspy.music.Music*, *measure\_resolution*: int) → float

Return the ratio of empty measures.

The empty-measure rate is defined as the ratio of the number of empty measures (where no note is played) to the total number of measures. Note that this metric only works for songs with a constant time signature. Return NaN if song length is zero. This metric is used in [1].

$$\text{empty\_measure\_rate} = \frac{\#(\text{empty\_measures})}{\#(\text{measures})}$$

**Parameters**

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure\_resolution** (*int*) – Time steps per measure.

**Returns** Empty-measure rate.

**Return type** *float*

See also:

*muspy.empty\_beat\_rate()* Compute the ratio of empty beats.

**References**

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

## 7.10 Technical Documentation

These are the detailed technical documentation.

### 7.10.1 muspy

A toolkit for symbolic music generation.

MusPy is an open source Python library for symbolic music generation. It provides essential tools for developing a music generation system, including dataset management, data I/O, data preprocessing and model evaluation.

**Features**

- Dataset management system for commonly used datasets with interfaces to PyTorch and TensorFlow.
- Data I/O for common symbolic music formats (e.g., MIDI, MusicXML and ABC) and interfaces to other symbolic music libraries (e.g., music21, mido, pretty\_midi and Pypianoroll).
- Implementations of common music representations for music generation, including the pitch-based, the event-based, the piano-roll and the note-based representations.
- Model evaluation tools for music generation systems, including audio rendering, score and piano-roll visualizations and objective metrics.

**class** *muspy.Base* (*\*\*kwargs*)

Base class for MusPy classes.

This is the base class for MusPy classes. It provides two handy I/O methods—*from\_dict* and *to\_ordered\_dict*. It also provides intuitive *\_\_repr\_\_* as well as methods *pretty\_str* and *print* for beautifully printing the content.

**Hint:** To implement a new class in MusPy, please inherit from this class and set the following class variables properly.

- `_attributes`: An `OrderedDict` with attribute names as keys and their types as values.
- `_optional_attributes`: A list of optional attribute names.
- `_list_attributes`: A list of attributes that are lists.

Take `muspy.Note` for example.:

```
_attributes = OrderedDict(
    [
        ("time", int),
        ("duration", int),
        ("pitch", int),
        ("velocity", int),
        ("pitch_str", str),
    ]
)
_optional_attributes = ["pitch_str"]
```

See also:

**`muspy.ComplexBase`** Base class that supports advanced operations on list attributes.

**`adjust_time`** (*func*: `Callable[[int], int]`, *attr*: `Optional[str] = None`, *recursive*: `bool = True`) → `BaseType`  
Adjust the timing of time-stamped objects.

#### Parameters

- **`func`** (*callable*) – The function used to compute the new timing from the old timing, i.e., `new_time = func(old_time)`.
- **`attr`** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **`recursive`** (*bool*) – Whether to apply recursively. Defaults to `True`.

#### Returns

**Return type** Object itself.

**`copy`** () → `BaseType`  
Return a shallow copy of the object.

This is equivalent to `copy.copy(self)()`.

#### Returns

**Return type** Shallow copy of the object.

**`deepcopy`** () → `BaseType`  
Return a deep copy of the object.

This is equivalent to `copy.deepcopy(self)()`

#### Returns

**Return type** Deep copy of the object.

**classmethod** `from_dict` (*dict\_*: *Mapping*[*KT*, *VT\_co*]) → *BaseType*

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

**Parameters** `dict` (*dict* or *mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., {“attr1”: *value1*, “attr2”: *value2*}.

**Returns**

**Return type** Constructed object.

**is\_valid** (*attr*: *Optional*[*str*] = *None*, *recursive*: *bool* = *True*) → *bool*

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns** Whether the attribute has a valid type and value.

**Return type** *bool*

**See also:**

*muspy.Base.validate()* Raise an error if an attribute has an invalid type or value.

*muspy.Base.is\_valid\_type()* Return True if an attribute is of a valid type.

**is\_valid\_type** (*attr*: *Optional*[*str*] = *None*, *recursive*: *bool* = *True*) → *bool*

Return True if an attribute is of a valid type.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**See also:**

*muspy.Base.validate\_type()* Raise an error if a certain attribute is of an invalid type.

*muspy.Base.is\_valid()* Return True if an attribute has a valid type and value.

**pretty\_str** (*skip\_missing*: *bool* = *True*) → *str*

Return the attributes as a string in a YAML-like format.

**Parameters** `skip_missing` (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**Returns** Stored data as a string in a YAML-like format.

**Return type** *str*

**See also:**



**`muspy.Base.print()`** Print the attributes in a YAML-like format.

**`print(skip_missing: bool = True)`**

Print the attributes in a YAML-like format.

**Parameters** **`skip_missing`** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

**See also:**

**`muspy.Base.pretty_str()`** Return the the attributes as a string in a YAML-like format.

**`to_ordered_dict(skip_missing: bool = True, deepcopy: bool = True)`** → `collections.OrderedDict`

Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

**Parameters**

- **`skip_missing`** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **`deepcopy`** (*bool*) – Whether to make deep copies of the attributes. Defaults to True.

**Returns** A dictionary that stores the attributes and their values as key-value pairs, e.g., `{“attr1”: value1, “attr2”: value2}`.

**Return type** `OrderedDict`

**`validate(attr: Optional[str] = None, recursive: bool = True)`** → `BaseType`

Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **`attr`** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **`recursive`** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**See also:**

**`muspy.Base.is_valid()`** Return True if an attribute has a valid type and value.

**`muspy.Base.validate_type()`** Raise an error if an attribute is of an invalid type.

**`validate_type(attr: Optional[str] = None, recursive: bool = True)`** → `BaseType`

Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute’s attributes.

**Parameters**

- **`attr`** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **`recursive`** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**See also:**

**`muspy.Base.is_valid_type()`** Return True if an attribute is of a valid type.

**`muspy.Base.validate()`** Raise an error if an attribute has an invalid type or value.

**`class muspy.ComplexBase`** (*\*\*kwargs*)

Base class that supports advanced operations on list attributes.

This class extend the Base class with advanced operations on list attributes, including *append*, *remove\_invalid*, *remove\_duplicate* and *sort*.

**See also:**

**`muspy.Base`** Base class for MusPy classes.

**`append(obj)`** → `ComplexBaseType`

Append an object to the corresponding list.

This will automatically determine the list attributes to append based on the type of the object.

**Parameters** *obj* – Object to append.

**`extend(other: Union[ComplexBaseType, Iterable[T_co]], deepcopy: bool = False)`** → `ComplexBaseType`

Extend the list(s) with another object or iterable.

**Parameters**

- ***other*** (*muspy.ComplexBase* or iterable) – If an object of the same type is given, extend the list attributes with the corresponding list attributes of the other object. If an iterable is given, call *muspy.ComplexBase.append()* for each item.
- ***deepcopy*** (*bool*) – Whether to make deep copies of the appended objects. Defaults to False.

**Returns**

**Return type** Object itself.

**`remove_duplicate(attr: Optional[str] = None, recursive: bool = True)`** → `ComplexBaseType`

Remove duplicate items from a list attribute.

**Parameters**

- ***attr*** (*str*) – Attribute to check. Defaults to check all attributes.
- ***recursive*** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**`remove_invalid(attr: Optional[str] = None, recursive: bool = True)`** → `ComplexBaseType`

Remove invalid items from a list attribute.

**Parameters**

- ***attr*** (*str*) – Attribute to validate. Defaults to validate all attributes.
- ***recursive*** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**`sort(attr: Optional[str] = None, recursive: bool = True)`** → `ComplexBaseType`

Sort a list attribute.

**Parameters**

- **attr** (*str*) – Attribute to sort. Defaults to sort all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**class** `muspy.Annotation` (*time: int, annotation: Any, group: Optional[str] = None*)

A container for annotation.

**time**

Start time of the annotation, in time steps or seconds.

**Type** `int`

**annotation**

Annotation of any type.

**Type** `any`

**group**

Group name for better organizing the annotations.

**Type** `str`, optional

**class** `muspy.Chord` (*time: int, pitches: List[int], duration: int, velocity: Optional[int] = None, pitches\_str: Optional[List[int]] = None*)

A container for chord.

**time**

Start time of the chord, in time steps.

**Type** `int`

**pitches**

Note pitches, as MIDI note numbers.

**Type** `list of int`

**duration**

Duration of the chord, in time steps.

**Type** `int`

**velocity**

Chord velocity. Defaults to `muspy.DEFAULT_VELOCITY`.

**Type** `int`, optional

**pitches\_str**

Note pitches as strings, useful for distinguishing C# and Db.

**Type** `list of str`

**adjust\_time** (*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → `muspy.classes.Chord`

Adjust the timing of the chord.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., `new_time = func(old_time)`.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.

- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**clip** (*lower: int = 0, upper: int = 127*) → `muspy.classes.Chord`

Clip the velocity of the chord.

**Parameters**

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

**Returns**

**Return type** Object itself.

**end**

End time of the chord.

**start**

Start time of the chord.

**transpose** (*semitone: int*) → `muspy.classes.Chord`

Transpose the notes by a number of semitones.

**Parameters** **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

**Returns**

**Return type** Object itself.

**class** `muspy.KeySignature` (*time: int, root: Optional[int] = None, mode: Optional[str] = None, fifths: Optional[int] = None, root\_str: Optional[str] = None*)

A container for key signature.

**time**

Start time of the key signature, in time steps or seconds.

**Type** `int`

**root**

Root (tonic) of the key signature.

**Type** `int`, optional

**mode**

Mode of the key signature.

**Type** `str`, optional

**fifths**

Number of sharps or flats. Positive numbers for sharps and negative numbers for flats.

**Type** `int`, optional

**root\_str**

Root of the key signature as a string.

**Type** `str`, optional

**class** `muspy.Lyric` (*time: int, lyric: str*)

A container for lyric.

---

```

time
    Start time of the lyric, in time steps or seconds.
        Type int

lyric
    Lyric (sentence, word, syllable, etc.).
        Type str

class muspy.Metadata (schema_version: str = '0.0', title: Optional[str] = None, creators: Op-
                        tional[List[str]] = None, copyright: Optional[str] = None, collection: Op-
                        tional[str] = None, source_filename: Optional[str] = None, source_format:
                        Optional[str] = None)
    A container for metadata.

    schema_version
        Schema version. Defaults to the latest version.
            Type str

    title
        Song title.
            Type str, optional

    creators
        Creator(s) of the song.
            Type list of str, optional

    copyright
        Copyright notice.
            Type str, optional

    collection
        Name of the collection.
            Type str, optional

    source_filename
        Name of the source file.
            Type str, optional

    source_format
        Format of the source file.
            Type str, optional

class muspy.Note (time: int, pitch: int, duration: int, velocity: Optional[int] = None, pitch_str: Op-
                    tional[str] = None)
    A container for note.

    time
        Start time of the note, in time steps.
            Type int

    pitch
        Note pitch, as a MIDI note number.
            Type int

```

**duration**

Duration of the note, in time steps.

**Type** `int`

**velocity**

Note velocity. Defaults to `muspy.DEFAULT_VELOCITY`.

**Type** `int`, optional

**pitch\_str**

Note pitch as a string, useful for distinguishing C# and Db.

**Type** `str`

**adjust\_time** (*func*: Callable[[`int`], `int`], *attr*: Optional[`str`] = None, *recursive*: bool = True) → muspy.classes.Note

Adjust the timing of the note.

**Parameters**

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., `new_time = func(old_time)`.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

**Returns**

**Return type** Object itself.

**clip** (*lower*: `int` = 0, *upper*: `int` = 127) → muspy.classes.Note

Clip the velocity of the note.

**Parameters**

- **lower** (*int*, *optional*) – Lower bound. Defaults to 0.
- **upper** (*int*, *optional*) – Upper bound. Defaults to 127.

**Returns**

**Return type** Object itself.

**end**

End time of the note.

**start**

Start time of the note.

**transpose** (*semitone*: `int`) → muspy.classes.Note

Transpose the note by a number of semitones.

**Parameters** **semitone** (*int*) – Number of semitones to transpose the note. A positive value raises the pitch, while a negative value lowers the pitch.

**Returns**

**Return type** Object itself.

**class** `muspy.Tempo` (*time*: `int`, *qpm*: `float`)

A container for key signature.

**time**

Start time of the tempo, in time steps.

**Type** `int`

**qpm**

Tempo in qpm (quarters per minute).

**Type** `float`**class** `muspy.TimeSignature` (*time: int, numerator: int, denominator: int*)

A container for time signature.

**time**

Start time of the time signature, in time steps or seconds.

**Type** `int`**numerator**

Numerator of the time signature.

**Type** `int`**denominator**

Denominator of the time signature.

**Type** `int`

**class** `muspy.Track` (*program: int = 0, is\_drum: bool = False, name: Optional[str] = None, notes: Optional[List[muspy.classes.Note]] = None, chords: Optional[List[muspy.classes.Chord]] = None, lyrics: Optional[List[muspy.classes.Lyric]] = None, annotations: Optional[List[muspy.classes.Annotation]] = None*)

A container for music track.

**program**Program number according to General MIDI specification<sup>1</sup>. Defaults to 0 (Acoustic Grand Piano).**Type** `int`, 0-127, optional**is\_drum**

Whether it is a percussion track. Defaults to False.

**Type** `bool`, optional**name**

Track name.

**Type** `str`, optional**notes**

Musical notes. Defaults to an empty list.

**Type** list of `muspy.Note`, optional**chords**

Chords. Defaults to an empty list.

**Type** list of `muspy.Chord`, optional**annotations**

Annotations. Defaults to an empty list.

**Type** list of `muspy.Annotation`, optional**lyrics**

Lyrics. Defaults to an empty list.

**Type** list of `muspy.Lyric`, optional<sup>1</sup> <https://www.midi.org/specifications/item/gm-level-1-sound-set>

---

**Note:** Indexing a Track object returns the note at a certain index. That is, `track[idx]` returns `track.notes[idx]`. Length of a Track object is the number of notes. That is, `len(track)` returns `len(track.notes)`.

---

## References

**clip** (*lower: int = 0, upper: int = 127*) → `muspy.classes.Track`  
Clip the velocity of each note.

### Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

### Returns

**Return type** Object itself.

**get\_end\_time** (*is\_sorted: bool = False*) → `int`  
Return the time of the last event.

This includes notes, chords, lyrics and annotations.

**Parameters** **is\_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

**transpose** (*semitone: int*) → `muspy.classes.Track`  
Transpose the notes by a number of semitones.

**Parameters** **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

### Returns

**Return type** Object itself.

`muspy.adjust_resolution` (*music: muspy.music.Music, target: Optional[int] = None, factor: Optional[float] = None, rounding: Union[str, Callable, None] = 'round'*) → `muspy.music.Music`  
Adjust resolution and timing of all time-stamped objects.

### Parameters

- **music** (*muspy.Music*) – Object to adjust the resolution.
- **target** (*int, optional*) – Target resolution.
- **factor** (*int or float, optional*) – Factor used to adjust the resolution based on the formula:  $new\_resolution = old\_resolution * factor$ . For example, a factor of 2 double the resolution, and a factor of 0.5 halve the resolution.
- **rounding** (*{'round', 'ceil', 'floor'} or callable, optional*) – Rounding mode. Defaults to 'round'.

`muspy.adjust_time` (*obj: muspy.base.Base, func: Callable[[int], int]*) → `muspy.base.Base`  
Adjust the timing of time-stamped objects.

### Parameters

- **obj** (*muspy.Music or muspy.Track*) – Object to adjust the timing.



- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new\_time = func(old\_time)*.

See also:

*muspy.adjust\_resolution()* Adjust the resolution and the timing of time-stamped objects.

---

**Note:** The resolution are left unchanged.

---

*muspy.append(obj1: muspy.base.ComplexBase, obj2) → muspy.base.ComplexBase*  
Append an object to the corresponding list.

This will automatically determine the list attributes to append based on the type of the object.

#### Parameters

- **obj1** (*muspy.ComplexBase*) – Object to which *obj2* to append.
- **obj2** – Object to be appended to *obj1*.

#### Notes

- If *obj1* is of type *muspy.Music*, *obj2* can be *muspy.Tempo*, *muspy.KeySignature*, *muspy.TimeSignature*, *muspy.Lyric*, *muspy.Annotation* or *muspy.Track*.
- If *obj1* is of type *muspy.Track*, *obj2* can be *muspy.Note*, *muspy.Chord*, *muspy.Lyric* or *muspy.Annotation*.

See also:

*muspy.ComplexBase.append* Equivalent function.

*muspy.clip(obj: Union[muspy.music.Music, muspy.classes.Track, muspy.classes.Note], lower: int = 0, upper: int = 127) → Union[muspy.music.Music, muspy.classes.Track, muspy.classes.Note]*  
Clip the velocity of each note.

#### Parameters

- **obj** (*muspy.Music*, *muspy.Track* or *muspy.Note*) – Object to clip.
- **lower** (*int* or *float*, *optional*) – Lower bound. Defaults to 0.
- **upper** (*int* or *float*, *optional*) – Upper bound. Defaults to 127.

*muspy.get\_end\_time(obj: Union[muspy.music.Music, muspy.classes.Track], is\_sorted: bool = False) → int*  
Return the the time of the last event in all tracks.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations.

#### Parameters

- **obj** (*muspy.Music* or *muspy.Track*) – Object to inspect.
- **is\_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

*muspy.get\_real\_end\_time(music: muspy.music.Music, is\_sorted: bool = False) → float*  
Return the end time in realtime.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations. Assume 120 qpm (quarter notes per minute) if no tempo information is available.

**Parameters**

- **music** (*muspy.Music*) – Object to inspect.
- **is\_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

`muspy.remove_duplicate` (*obj: muspy.base.ComplexBase*) → *muspy.base.ComplexBase*

Remove duplicate change events.

**Parameters** *obj* (*muspy.Music*) – Object to process.

`muspy.sort` (*obj: muspy.base.ComplexBase*) → *muspy.base.ComplexBase*

Sort all the time-stamped objects with respect to event time.

- If a *muspy.Music* is given, this will sort key signatures, time signatures, lyrics and annotations, along with notes, lyrics and annotations for each track.
- If a *muspy.Track* is given, this will sort notes, lyrics and annotations.

**Parameters** *obj* (*muspy.ComplexBase*) – Object to sort.

`muspy.to_ordered_dict` (*obj: muspy.base.Base, skip\_missing: bool = True, deepcopy: bool = True*) → *collections.OrderedDict*

Return an OrderedDict converted from a Music object.

**Parameters**

- **obj** (*muspy.Base*) – Object to convert.
- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **deepcopy** (*bool*) – Whether to make deep copies of the attributes. Defaults to False.

**Returns** Converted OrderedDict.

**Return type** *OrderedDict*

`muspy.transpose` (*obj: Union[muspy.music.Music, muspy.classes.Track, muspy.classes.Note], semitone: int*) → *Union[muspy.music.Music, muspy.classes.Track, muspy.classes.Note]*

Transpose all the notes by a number of semitones.

**Parameters**

- **obj** (*muspy.Music, muspy.Track* or *muspy.Note*) – Object to transpose.
- **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

**class** `muspy.ABCFolderDataset` (*root: Union[str, pathlib.Path], convert: Optional[bool] = None, kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: Optional[bool] = None, use\_converted: Optional[bool] = None*)

Class for datasets storing ABC files in a folder.

**See also:**

*muspy.FolderDataset* Class for datasets storing files in a folder.

`on_the_fly` () → *FolderDatasetType*

Enable on-the-fly mode and convert the data on the fly.

**Returns**

**Return type** Object itself.

**read** (*filename: Tuple[str, Tuple[int, int]]*) → *muspy.music.Music*

Read a file into a Music object.

**class** *muspy.Dataset*

Base class for MusPy datasets.

To build a custom dataset, it should inherit this class and override the methods `__getitem__` and `__len__` as well as the class attribute `_info`. `__getitem__` should return the *i*-th data sample as a *muspy.Music* object. `__len__` should return the size of the dataset. `_info` should be a *muspy.DatasetInfo* instance storing the dataset information.

**classmethod** *citation* ()

Print the citation information.

**classmethod** *info* ()

Return the dataset information.

**save** (*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n\_jobs: int = 1, ignore\_exceptions:*

*bool = True, verbose: bool = True, \*\*kwargs*)

Save all the music objects to a directory.

#### Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to *muspy.save\_json()* or *muspy.save\_yaml()*

#### Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**split** (*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None*) → *Dict[str, List[int]]*

Return the dataset as a PyTorch dataset.

#### Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to *numpy*.

`random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or Dict of `:class:torch.utils.data.Dataset`

**to\_tensorflow\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If int or array\_like, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**Returns**

- `class:tensorflow.data.Dataset` or Dict of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

**class** `muspy.DatasetInfo` (*name: Optional[str] = None, description: Optional[str] = None, homepage: Optional[str] = None, license: Optional[str] = None*)

A container for dataset information.

**class** `muspy.EssenFolkSongDatabase` (*root: Union[str, pathlib.Path], download\_and\_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: Optional[bool] = None, use\_converted: Optional[bool] = None, verbose: bool = True*)

Essen Folk Song Database.

**class** `muspy.FolderDataset` (*root: Union[str, pathlib.Path], convert: Optional[bool] = None, kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: Optional[bool] = None, use\_converted: Optional[bool] = None*)

Class for datasets storing files in a folder.

This class extends `muspy.Dataset` to support folder datasets. To build a custom folder dataset, please refer to the documentation of `muspy.Dataset` for details. In addition, set class attribute `_extension` to the extension to look for when building the dataset and set `read` to a callable that takes as inputs a filename of a source file and return the converted Music object.

**root**

Root directory of the dataset.

Type `str` or `Path`

### Parameters

- **convert** (*bool, optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use\_converted** (*bool, optional*) – Force to disable on-the-fly mode and use stored converted data

---

**Important:** `muspy.FolderDataset.converted_exists()` depends solely on a special file named `.muspy.success` in the folder `{root}/_converted/`, which serves as an indicator for the existence and integrity of the converted dataset. If the converted dataset is built by `muspy.FolderDataset.convert()`, the `.muspy.success` file will be created as well. If the converted dataset is created manually, make sure to create the `.muspy.success` file in the folder `{root}/_converted/` to prevent errors.

---

## Notes

Two modes are available for this dataset. When the on-the-fly mode is enabled, a data sample is converted to a music object on the fly when being indexed. When the on-the-fly mode is disabled, a data sample is loaded from the precomputed converted data.

See also:

***muspy.Dataset*** Base class for MusPy datasets.

**convert** (*kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: bool = True, verbose: bool = True*) → FolderDatasetType  
Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

### Parameters

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.

### Returns

**Return type** Object itself.

**converted\_dir**

Path to the root directory of the converted dataset.

**converted\_exists** () → bool

Return True if the saved dataset exists, otherwise False.

**exists** () → bool

Return True if the dataset exists, otherwise False.

**load** (*filename: Union[str, pathlib.Path]*) → *muspy.music.Music*

Load a file into a Music object.

**on\_the\_fly** () → FolderDatasetType

Enable on-the-fly mode and convert the data on the fly.

### Returns

**Return type** Object itself.

**read** (*filename: Any*) → *muspy.music.Music*

Read a file into a Music object.

**use\_converted** () → FolderDatasetType

Disable on-the-fly mode and use converted data.

### Returns

**Return type** Object itself.

```
class muspy.HaydnOp20Dataset (root: Union[str, pathlib.Path], download_and_extract: bool =
                             False, overwrite: Optional[bool] = None, cleanup: Optional[bool]
                             = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs:
                             int = 1, ignore_exceptions: Optional[bool] = None, use_converted:
                             Optional[bool] = None, verbose: bool = True)
```

Haydn Op.20 Dataset.

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.HymnalDataset (root: Union[str, pathlib.Path], download: bool = False, convert: bool
                           = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool =
                           True, use_converted: Optional[bool] = None)
```

Hymnal Dataset.

```
download () → muspy.datasets.base.FolderDataset
    Download the source datasets.
```

**Returns**

**Return type** Object itself.

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.HymnalTuneDataset (root: Union[str, pathlib.Path], download: bool = False, con-
                               vert: bool = False, kind: str = 'json', n_jobs: int = 1, ig-
                               nore_exceptions: bool = True, use_converted: Optional[bool] =
                               None)
```

Hymnal Dataset (tune only).

```
download () → muspy.datasets.base.FolderDataset
    Download the source datasets.
```

**Returns**

**Return type** Object itself.

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.JSBChoralesDataset (root: Union[str, pathlib.Path], download_and_extract: bool
                                = False, overwrite: Optional[bool] = None, cleanup: Op-
                                tional[bool] = None, convert: Optional[bool] = None, kind:
                                str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] =
                                None, use_converted: Optional[bool] = None, verbose: bool =
                                True)
```

Johann Sebastian Bach Chorales Dataset.

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.LakhMIDIAlignedDataset (root: Union[str, pathlib.Path], download_and_extract:
                                    bool = False, overwrite: Optional[bool] = None, cleanup:
                                    Optional[bool] = None, convert: Optional[bool] = None,
                                    kind: str = 'json', n_jobs: int = 1, ignore_exceptions:
                                    Optional[bool] = None, use_converted: Optional[bool] =
                                    None, verbose: bool = True)
```

Lakh MIDI Dataset - aligned subset.

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.LakhMIDIDataset (root: Union[str, pathlib.Path], download_and_extract: bool = False,
                             overwrite: Optional[bool] = None, cleanup: Optional[bool] = None,
                             convert: Optional[bool] = None, kind: str = 'json', n_jobs: int =
                             1, ignore_exceptions: Optional[bool] = None, use_converted: Op-
                             tional[bool] = None, verbose: bool = True)
```

Lakh MIDI Dataset.

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.LakhMIDIMatchedDataset (root: Union[str, pathlib.Path], download_and_extract:
                                     bool = False, overwrite: Optional[bool] = None, cleanup:
                                     Optional[bool] = None, convert: Optional[bool] = None,
                                     kind: str = 'json', n_jobs: int = 1, ignore_exceptions:
                                     Optional[bool] = None, use_converted: Optional[bool] =
                                     None, verbose: bool = True)
```

Lakh MIDI Dataset - matched subset.

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.MAESTRODatasetV1 (root: Union[str, pathlib.Path], download_and_extract: bool =
                               False, overwrite: Optional[bool] = None, cleanup: Optional[bool]
                               = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs:
                               int = 1, ignore_exceptions: Optional[bool] = None, use_converted:
                               Optional[bool] = None, verbose: bool = True)
```

MAESTRO Dataset (MIDI only).

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.MAESTRODatasetV2 (root: Union[str, pathlib.Path], download_and_extract: bool =
                               False, overwrite: Optional[bool] = None, cleanup: Optional[bool]
                               = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs:
                               int = 1, ignore_exceptions: Optional[bool] = None, use_converted:
                               Optional[bool] = None, verbose: bool = True)
```

MAESTRO Dataset (MIDI only).

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.Music21Dataset (composer: Optional[str] = None)
    A class of datasets containing files in music21 corpus.
```

#### Parameters

- **composer** (*str*) – Name of a composer or a collection. Please refer to the music21 corpus reference page for a full list [1].
- **extensions** (*list of str*) – File extensions of desired files.

#### References

[1] <https://web.mit.edu/music21/doc/about/referenceCorpus.html>

```
convert (root: Union[str, pathlib.Path], kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool =
          True) → muspy.datasets.base.MusicDataset
    Convert and save the Music objects.
```

#### Parameters



- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

**class** muspy.**MusicDataset** (*root: Union[str, pathlib.Path], kind: str = 'json'*)

Class for datasets of MusPy JSON/YAML files.

**root**

Root directory of the dataset.

**Type** *str or Path*

**kind**

File format of the data. Defaults to 'json'.

**Type** *{'json', 'yaml'}, optional*

**See also:**

*muspy.Dataset* Base class for MusPy datasets.

**class** muspy.**MusicNetDataset** (*root: Union[str, pathlib.Path], download\_and\_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: Optional[bool] = None, use\_converted: Optional[bool] = None, verbose: bool = True*)

MusicNet Dataset (MIDI only).

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music

Read a file into a Music object.

**class** muspy.**NESMusicDatabase** (*root: Union[str, pathlib.Path], download\_and\_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: Optional[bool] = None, use\_converted: Optional[bool] = None, verbose: bool = True*)

NES Music Database.

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music

Read a file into a Music object.

**class** muspy.**NottinghamDatabase** (*root: Union[str, pathlib.Path], download\_and\_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: Optional[bool] = None, use\_converted: Optional[bool] = None, verbose: bool = True*)

Nottingham Database.

```
class muspy.RemoteABCFolderDataset (root: Union[str, pathlib.Path], download_and_extract:
    bool = False, overwrite: Optional[bool] = None, cleanup:
    Optional[bool] = None, convert: Optional[bool] = None,
    kind: str = 'json', n_jobs: int = 1, ignore_exceptions:
    Optional[bool] = None, use_converted: Optional[bool] =
    None, verbose: bool = True)
```

Base class for remote datasets storing ABC files in a folder.

**See also:**

[`muspy.ABCFolderDataset`](#) Class for datasets storing ABC files in a folder.

[`muspy.RemoteDataset`](#) Base class for remote MusPy datasets.

```
class muspy.RemoteDataset (root: Union[str, pathlib.Path], download_and_extract: Optional[bool]
    = None, overwrite: Optional[bool] = None, cleanup: Optional[bool] =
    None, verbose: bool = True)
```

Base class for remote MusPy datasets.

This class extends [`muspy.Dataset`](#) to support remote datasets. To build a custom remote dataset, please refer to the documentation of [`muspy.Dataset`](#) for details. In addition, set the class attribute `_sources` to the URLs to the source files (see Notes).

**root**

Root directory of the dataset.

**Type** `str` or `Path`

#### Parameters

- **download\_and\_extract** (*bool*, *optional*) – Whether to download and extract the dataset. Defaults to `False`.
- **overwrite** (*bool*, *optional*) – Whether to overwrite existing downloaded files. Defaults to `True`.
- **cleanup** (*bool*, *optional*) – Whether to remove the source archive(s). Defaults to `False`.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to `True`.

**Raises** `RuntimeError`: – If `download_and_extract` is `False` but file `{root}/.muspy.success` does not exist (see below).

---

**Important:** `muspy.Dataset.exists()` depends solely on a special file named `.muspy.success` in directory `{root}/_converted/`. This file serves as an indicator for the existence and integrity of the dataset. It will automatically be created if the dataset is successfully downloaded and extracted by `muspy.Dataset.download_and_extract()`. If the dataset is downloaded manually, make sure to create the `.muspy.success` file in directory `{root}/_converted/` to prevent errors.

---

#### Notes

The class attribute `_sources` is a dictionary storing the following information of each source file.

- `filename` (`str`): Name to save the file.
- `url` (`str`): URL to the file.

- **archive** (bool): Whether the file is an archive.
- **md5** (str, optional): Expected MD5 checksum of the file.

Here is an example.:

```
_sources = {
    "example": {
        "filename": "example.tar.gz",
        "url": "https://www.example.com/example.tar.gz",
        "archive": True,
        "md5": None,
    }
}
```

See also:

***muspy.Dataset*** Base class for MusPy datasets.

**download** (*overwrite: bool = True, verbose: bool = True*) → RemoteDatasetType

Download the source datasets.

#### Parameters

- **overwrite** (*bool, optional*) – Whether to overwrite existing files. Defaults to True.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.

#### Returns

**Return type** Object itself.

**download\_and\_extract** (*overwrite: bool = True, cleanup: bool = False, verbose: bool = True*) → RemoteDatasetType

Download source datasets and extract the downloaded archives.

#### Parameters

- **overwrite** (*bool, optional*) – Whether to overwrite existing files. Defaults to True.
- **cleanup** (*bool, optional*) – Whether to remove the source archive. Defaults to False.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.

#### Returns

**Return type** Object itself.

**exists** () → bool

Return True if the dataset exists, otherwise False.

**extract** (*cleanup: bool = False, verbose: bool = True*) → RemoteDatasetType

Extract the downloaded archive(s).

#### Parameters

- **cleanup** (*bool, optional*) – Whether to remove the source archive after extraction. Defaults to False.
- **verbose** (*bool, optional*) – Whether to be verbose. Defaults to True.

#### Returns

**Return type** Object itself.

**source\_exists**( ) → bool

Return True if all the sources exist, otherwise False.

```
class muspy.RemoteFolderDataset (root: Union[str, pathlib.Path], download_and_extract: bool
                                = False, overwrite: Optional[bool] = None, cleanup: Op-
                                tional[bool] = None, convert: Optional[bool] = None, kind: str
                                = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] =
                                None, use_converted: Optional[bool] = None, verbose: bool =
                                True)
```

Base class for remote datasets storing files in a folder.

**root**

Root directory of the dataset.

**Type** str or Path

#### Parameters

- **download\_and\_extract** (bool, optional) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (bool, optional) – Whether to remove the source archive(s). Defaults to False.
- **convert** (bool, optional) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** ({'json', 'yaml'}, optional) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (int, optional) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (bool, optional) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use\_converted** (bool, optional) – Force to disable on-the-fly mode and use stored converted data

**See also:**

**[muspy.FolderDataset](#)** Class for datasets storing files in a folder.

**[muspy.RemoteDataset](#)** Base class for remote MusPy datasets.

**read** (filename: str) → muspy.music.Music

Read a file into a Music object.

```
class muspy.RemoteMusicDataset (root: Union[str, pathlib.Path], download_and_extract: bool
                                = False, overwrite: Optional[bool] = None, cleanup: Op-
                                tional[bool] = None, kind: str = 'json', verbose: bool = True)
```

Base class for remote datasets of MusPy JSON/YAML files.

**root**

Root directory of the dataset.

**Type** str or Path

**kind**

File format of the data. Defaults to 'json'.

**Type** {'json', 'yaml'}, optional

**Parameters**

- **download\_and\_extract** (*bool*, *optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool*, *optional*) – Whether to remove the source archive(s). Defaults to False.

**See also:**

***muspy.MusicDataset*** Class for datasets of MusPy JSON/YAML files.

***muspy.RemoteDataset*** Base class for remote MusPy datasets.

```
class muspy.WikifoniaDataset (root: Union[str, pathlib.Path], download_and_extract: bool =
                             False, overwrite: Optional[bool] = None, cleanup: Optional[bool]
                             = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs:
                             int = 1, ignore_exceptions: Optional[bool] = None, use_converted:
                             Optional[bool] = None, verbose: bool = True)
```

Wikifonia dataset.

**read** (filename: Union[str, pathlib.Path]) → muspy.music.Music  
Read a file into a Music object.

**muspy.get\_dataset** (key: str) → Type[muspy.datasets.base.Dataset]  
Return a certain dataset class by key.

**Parameters** **key** (*str*) – Dataset key (case-insensitive).

**Returns**

**Return type** The corresponding dataset class.

**muspy.list\_datasets** ()  
Return all supported dataset classes as a list.

**Returns**

**Return type** A list of all supported dataset classes.

**muspy.download\_bravura\_font** ()  
Download the Bravura font.

**muspy.download\_musescore\_soundfont** ()  
Download the MuseScore General soundfont.

**muspy.get\_bravura\_font\_dir** () → pathlib.Path  
Return path to the directory of the Bravura font.

**muspy.get\_bravura\_font\_path** () → pathlib.Path  
Return path to the Bravura font.

**muspy.get\_musescore\_soundfont\_dir** () → pathlib.Path  
Return path to the MuseScore General soundfont directory.

**muspy.get\_musescore\_soundfont\_path** () → pathlib.Path  
Return path to the MuseScore General soundfont.

**exception** `muspy.MIDIError`

An error class for MIDI related exceptions.

**exception** `muspy.MusicXMLError`

An error class for MusicXML related exceptions.

`muspy.from_event_representation` (*array: numpy.ndarray, resolution: int = 24, program: int = 0, is\_drum: bool = False, use\_single\_note\_off\_event: bool = False, use\_end\_of\_sequence\_event: bool = False, max\_time\_shift: int = 100, velocity\_bins: int = 32, default\_velocity: int = 64, duplicate\_note\_mode: str = 'fifo'*) → `muspy.music.Music`

Decode event-based representation into a Music object.

#### Parameters

- **array** (*ndarray*) – Array in event-based representation to decode. Cast to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_single\_note\_off\_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, a note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use\_end\_of\_sequence\_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **max\_time\_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than `max_time_shift` will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity\_bins** (*int*) – Number of velocity bins to use. Defaults to 32.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.
- **duplicate\_note\_mode** (*{'fifo', 'lifo', 'close\_all'}*) – Policy for dealing with duplicate notes. When a note off event is presetned while there are multiple corresponding note on events that have not yet been closed, we need a policy to decide which note on messages to close. This is only effective when `use_single_note_off_event` is False. Defaults to 'fifo'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out): close the latest note on
  - 'close\_all': close all note on messages

**Returns** Decoded Music object.

**Return type** `muspy.Music`

## References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

`muspy.from_mido` (*midi*: `mido.midifiles.MidiFile`, *duplicate\_note\_mode*: `str = 'fifo'`) → `muspy.music.Music`

Return a mido MidiFile object as a Music object.

### Parameters

- **midi** (`mido.MidiFile`) – Mido MidiFile object to convert.
- **duplicate\_note\_mode** (`{'fifo', 'lifo', 'close_all'}`) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspodng note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out): close the latest note on
  - 'close\_all': close all note on messages

**Returns** Converted Music object.

**Return type** `muspy.Music`

`muspy.from_music21` (*stream*: `music21.stream.Stream`, *resolution*=24) → Union[`muspy.music.Music`, List[`muspy.music.Music`], `muspy.classes.Track`, List[`muspy.classes.Track`]]

Return a music21 Stream object as Music or Track object(s).

### Parameters

- **stream** (`music21.stream.Stream`) – Stream object to convert.
- **resolution** (`int`, *optional*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Converted Music or Track object(s).

**Return type** `muspy.Music` or `muspy.Track`

`muspy.from_music21_opus` (*opus*: `music21.stream.Opus`, *resolution*=24) → List[`muspy.music.Music`]

Return a music21 Opus object as a list of Music objects.

### Parameters

- **opus** (`music21.stream.Opus`) – Opus object to convert.
- **resolution** (`int`, *optional*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Converted Music object.

**Return type** `muspy.Music`

`muspy.from_music21_part` (*part*: `music21.stream.Part`, *resolution*=24) → Union[`muspy.classes.Track`, List[`muspy.classes.Track`]]

Return a music21 Part object as Track object(s).

### Parameters

- **part** (`music21.stream.Part`) – Part object to parse.
- **resolution** (`int`, *optional*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Parsed track(s).

**Return type** `muspy.Track` or list of `muspy.Track`

`muspy.from_music21_score` (*score: music21.stream.Score, resolution=24*) → `muspy.music.Music`  
Return a music21 Stream object as a Music object.

**Parameters**

- **score** (*music21.stream.Score*) – Score object to convert.
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Converted Music object.

**Return type** `muspy.Music`

`muspy.from_note_representation` (*array: numpy.ndarray, resolution: int = 24, program: int = 0, is\_drum: bool = False, use\_start\_end: bool = False, encode\_velocity: bool = True, default\_velocity: int = 64*) → `muspy.music.Music`  
Decode note-based representation into a Music object.

**Parameters**

- **array** (*ndarray*) – Array in note-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_start\_end** (*bool*) – Whether to use ‘start’ and ‘end’ to encode the timing rather than ‘time’ and ‘duration’. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **default\_velocity** (*int*) – Default velocity value to use when decoding if *encode\_velocity* is False. Defaults to 64.

**Returns** Decoded Music object.

**Return type** `muspy.Music`

## References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

`muspy.from_object` (*obj: Union[music21.stream.Stream, mido.midifiles.midifiles.MidiFile, pretty\_midi.pretty\_midi.PrettyMIDI, pypianoroll.multitrack.Multitrack], \*\*kwargs*) → `Union[muspy.music.Music, List[muspy.music.Music], muspy.classes.Track, List[muspy.classes.Track]]`  
Return an outside object as a Music object.

**Parameters**

- **obj** – Object to convert. Supported objects are `music21.Stream`, `mido.MidiTrack`, `pretty_midi.PrettyMIDI`, and `pypianoroll.Multitrack` objects.



- **\*\*kwargs** – Keyword arguments to pass to `muspy.from_music21()`, `muspy.from_mido()`, `from_pretty_midi()` or `from_pypianoroll()`.

**Returns** Converted Music object.

**Return type** `muspy.Music`

`muspy.from_pianoroll_representation` (*array: numpy.ndarray, resolution: int = 24, program: int = 0, is\_drum: bool = False, encode\_velocity: bool = True, default\_velocity: int = 64*) → `muspy.music.Music`

Decode pitch-based representation into a Music object.

#### Parameters

- **array** (*ndarray*) – Array in piano-roll representation to decode. Will be casted to integer if not of integer type. If `encode_velocity` is True, will be casted to boolean if not of boolean type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode velocities. Defaults to True.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** Decoded Music object.

**Return type** `muspy.Music`

#### References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

`muspy.from_pitch_representation` (*array: numpy.ndarray, resolution: int = 24, program: int = 0, is\_drum: bool = False, use\_hold\_state: bool = False, default\_velocity: int = 64*) → `muspy.music.Music`

Decode pitch-based representation into a Music object.

#### Parameters

- **array** (*ndarray*) – Array in pitch-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_hold\_state** (*bool*) – Whether to use a special state for holds. Defaults to False.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** Decoded Music object.

**Return type** `muspy.Music`

## References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

`muspy.from_pretty_midi` (*midi*: `pretty_midi.pretty_midi.PrettyMIDI`, *resolution*: `Optional[int] = None`) → `muspy.music.Music`

Return a pretty\_midi PrettyMIDI object as a Music object.

### Parameters

- **midi** (`pretty_midi.PrettyMIDI`) – PrettyMIDI object to convert.
- **resolution** (*int*, *optional*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Converted Music object.

**Return type** `muspy.Music`

`muspy.from_pypianoroll` (*multitrack*: `pypianoroll.multitrack.Multitrack`, *default\_velocity*: *int* = 64) → `muspy.music.Music`

Return a Pypianoroll Multitrack object as a Music object.

### Parameters

- **multitrack** (`pypianoroll.Multitrack`) – Pypianoroll Multitrack object to convert.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** `music` – Converted MusPy Music object.

**Return type** `muspy.Music`

`muspy.from_pypianoroll_track` (*track*: `pypianoroll.track.Track`, *default\_velocity*: *int* = 64) → `muspy.classes.Track`

Return a Pypianoroll Track object as a Track object.

### Parameters

- **track** (`pypianoroll.Track`) – Pypianoroll Track object to convert.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** Converted track.

**Return type** `muspy.Track`

`muspy.from_representation` (*array*: `numpy.ndarray`, *kind*: *str*, *\*\*kwargs*) → `muspy.music.Music`

Update with the given representation.

### Parameters

- **array** (`numpy.ndarray`) – Array in a supported representation.
- **kind** (*str*, {'pitch', 'pianoroll', 'event', 'note'}) – Data representation.

- **\*\*kwargs** – Keyword arguments to pass to `muspy.from_pitch_representation()`, `muspy.from_pianoroll_representation()`, `muspy.from_event_representation()` or `muspy.from_note_representation()`.

**Returns** Converted Music object.

**Return type** `muspy.Music`

`muspy.load(path: Union[str, pathlib.Path, TextIO], kind: Optional[str] = None, **kwargs) → muspy.music.Music`  
Load a JSON or a YAML file into a Music object.

This is a wrapper function for `muspy.load_json()` and `muspy.load_yaml()`.

#### Parameters

- **path** (`str`, `Path` or `TextIO`) – Path to the file or the file to load.
- **kind** (`{'json', 'yaml'}`, `optional`) – Format to save. Defaults to infer from the extension.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.load_json()` or `muspy.load_yaml()`.

**Returns** Loaded Music object.

**Return type** `muspy.Music`

See also:

`muspy.load_json()` Load a JSON file into a Music object.

`muspy.load_yaml()` Load a YAML file into a Music object.

`muspy.read()` Read a MIDI/MusicXML/ABC file into a Music object.

`muspy.load_json(path: Union[str, pathlib.Path, TextIO], compressed: Optional[bool] = None) → muspy.music.Music`  
Load a JSON file into a Music object.

#### Parameters

- **path** (`str`, `Path` or `TextIO`) – Path to the file or the file to load.
- **compressed** (`bool`, `optional`) – Whether the file is a compressed JSON file (`.json.gz`). Has no effect when `path` is a file object. Defaults to infer from the extension (`.gz`).

**Returns** Loaded Music object.

**Return type** `muspy.Music`

#### Notes

When a path is given, assume UTF-8 encoding and gzip compression if `compressed=True`.

`muspy.load_yaml(path: Union[str, pathlib.Path, TextIO], compressed: Optional[bool] = None) → muspy.music.Music`  
Load a YAML file into a Music object.

#### Parameters

- **path** (`str`, `Path` or `TextIO`) – Path to the file or the file to load.

- **compressed** (*bool*, *optional*) – Whether the file is a compressed YAML file (.yaml.gz). Has no effect when *path* is a file object. Defaults to infer from the extension (.gz).

**Returns** Loaded Music object.

**Return type** *muspy.Music*

## Notes

When a path is given, assume UTF-8 encoding and gzip compression if *compressed=True*.

`muspy.read(path: Union[str, pathlib.Path], kind: Optional[str] = None, **kwargs) → Union[muspy.music.Music, List[muspy.music.Music]]`  
Read a MIDI/MusicXML/ABC file into a Music object.

### Parameters

- **path** (*str* or *Path*) – Path to the file to read.
- **kind** ({'midi', 'musicxml', 'abc'}, *optional*) – Format to save. Defaults to infer from the extension.
- **\*\*kwargs** – Keyword arguments to pass to *muspy.read\_midi()*, *muspy.read\_musicxml()* or *read\_abc()*.

**Returns** Converted Music object(s).

**Return type** *muspy.Music* or list of *muspy.Music*

See also:

*muspy.load()* Load a JSON or a YAML file into a Music object.

`muspy.read_abc(path: Union[str, pathlib.Path], number: Optional[int] = None, resolution=24) → Union[muspy.music.Music, List[muspy.music.Music]]`  
Return an ABC file into Music object(s) using music21 backend.

### Parameters

- **path** (*str* or *Path*) – Path to the ABC file to read.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the 'X:' field).
- **resolution** (*int*, *optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.

**Returns** Converted Music object(s).

**Return type** list of *muspy.Music*

`muspy.read_abc_string(data_str: str, number: Optional[int] = None, resolution=24) → Union[muspy.music.Music, List[muspy.music.Music]]`  
Read ABC data into Music object(s) using music21 backend.

### Parameters

- **data\_str** (*str*) – ABC data to parse.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the 'X:' field).
- **resolution** (*int*, *optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.

**Returns** Converted Music object(s).

**Return type** `muspy.Music`

`muspy.read_midi(path: Union[str, pathlib.Path], backend: str = 'mido', duplicate_note_mode: str = 'fifo') → muspy.music.Music`  
 Read a MIDI file into a Music object.

#### Parameters

- **path** (`str` or `Path`) – Path to the MIDI file to read.
- **backend** (`{'mido', 'pretty_midi'}`) – Backend to use.
- **duplicate\_note\_mode** (`{'fifo', 'lifo', 'close_all'}`) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspoding note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'. Only used when `backend='mido'`.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out):close the latest note on
  - 'close\_all': close all note on messages

**Returns** Converted Music object.

**Return type** `muspy.Music`

`muspy.read_musicxml(path: Union[str, pathlib.Path], compressed: Optional[bool] = None) → muspy.music.Music`  
 Read a MusicXML file into a Music object.

**Parameters** **path** (`str` or `Path`) – Path to the MusicXML file to read.

**Returns** Converted Music object.

**Return type** `muspy.Music`

## Notes

Grace notes and unpitched notes are not supported.

`muspy.drum_in_pattern_rate(music: muspy.music.Music, meter: str) → float`  
 Return the ratio of drum notes in a certain drum pattern.

The drum-in-pattern rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Only drum tracks are considered. Return NaN if no drum note is found. This metric is used in [1].

$$\text{drum\_in\_pattern\_rate} = \frac{\#(\text{drum\_notes\_in\_pattern})}{\#(\text{drum\_notes})}$$

#### Parameters

- **music** (`muspy.Music`) – Music object to evaluate.
- **meter** (`str`, `{'duple', 'triple'}`) – Meter of the drum pattern.

**Returns** Drum-in-pattern rate.

**Return type** `float`

See also:

`muspy.drum_pattern_consistency()` Compute the largest drum-in-pattern rate.

## References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.drum_pattern_consistency` (*music*: *muspy.music.Music*) → float

Return the largest drum-in-pattern rate.

The drum pattern consistency is defined as the largest drum-in-pattern rate over duple and triple meters. Only drum tracks are considered. Return NaN if no drum note is found.

$$\text{drum\_pattern\_consistency} = \max_{\text{meter}} \text{drum\_in\_pattern\_rate}(\text{meter})$$

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Drum pattern consistency.

**Return type** float

See also:

`muspy.drum_in_pattern_rate()` Compute the ratio of drum notes in a certain drum pattern.

`muspy.empty_beat_rate` (*music*: *muspy.music.Music*) → float

Return the ratio of empty beats.

The empty-beat rate is defined as the ratio of the number of empty beats (where no note is played) to the total number of beats. Return NaN if song length is zero. This metric is also implemented in Pypianoroll [1].

$$\text{empty\_beat\_rate} = \frac{\#(\text{empty\_beats})}{\#(\text{beats})}$$

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Empty-beat rate.

**Return type** float

See also:

`muspy.empty_measure_rate()` Compute the ratio of empty measures.

## References

1. Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang, “Pypianoroll: Open Source Python Package for Handling Multitrack Pianorolls,” in Late-Breaking Demos of the 18th International Society for Music Information Retrieval Conference (ISMIR), 2018.

`muspy.empty_measure_rate` (*music*: *muspy.music.Music*, *measure\_resolution*: int) → float

Return the ratio of empty measures.

The empty-measure rate is defined as the ratio of the number of empty measures (where no note is played) to the total number of measures. Note that this metric only works for songs with a constant time signature. Return NaN if song length is zero. This metric is used in [1].

$$\text{empty\_measure\_rate} = \frac{\#(\text{empty\_measures})}{\#(\text{measures})}$$

**Parameters**

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure\_resolution** (*int*) – Time steps per measure.

**Returns** Empty-measure rate.

**Return type** *float*

See also:

*muspy.empty\_beat\_rate()* Compute the ratio of empty beats.

**References**

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

*muspy.groove\_consistency* (*music: muspy.music.Music, measure\_resolution: int*) → *float*

Return the groove consistency.

The groove consistency is defined as the mean hamming distance of the neighboring measures.

$$groove\_consistency = 1 - \frac{1}{T-1} \sum_{i=1}^{T-1} d(G_i, G_{i+1})$$

Here,  $T$  is the number of measures,  $G_i$  is the binary onset vector of the  $i$ -th measure (a one at position that has an onset, otherwise a zero), and  $d(G, G')$  is the hamming distance between two vectors  $G$  and  $G'$ . Note that this metric only works for songs with a constant time signature. Return NaN if the number of measures is less than two. This metric is used in [1].

**Parameters**

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure\_resolution** (*int*) – Time steps per measure.

**Returns** Groove consistency.

**Return type** *float*

**References**

1. Shih-Lun Wu and Yi-Hsuan Yang, “The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures”, in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

*muspy.n\_pitch\_classes\_used* (*music: muspy.music.Music*) → *int*

Return the number of unique pitch classes used.

Drum tracks are ignored.

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Number of unique pitch classes used.

**Return type** *int*

See also:

`muspy.n_pitches_used()` Compute the number of unique pitches used.

`muspy.n_pitches_used(music: muspy.music.Music) → int`  
Return the number of unique pitches used.

Drum tracks are ignored.

**Parameters** `music` (`muspy.Music`) – Music object to evaluate.

**Returns** Number of unique pitch used.

**Return type** `int`

See also:

`muspy.n_pitch_class_used()` Compute the number of unique pitch classes used.

`muspy.pitch_class_entropy(music: muspy.music.Music) → float`  
Return the entropy of the normalized note pitch class histogram.

The pitch class entropy is defined as the Shannon entropy of the normalized note pitch class histogram. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$\text{pitch\_class\_entropy} = - \sum_{i=0}^{11} P(\text{pitch\_class} = i) \times \log_2 P(\text{pitch\_class} = i)$$

**Parameters** `music` (`muspy.Music`) – Music object to evaluate.

**Returns** Pitch class entropy.

**Return type** `float`

See also:

`muspy.pitch_entropy()` Compute the entropy of the normalized pitch histogram.

## References

1. Shih-Lun Wu and Yi-Hsuan Yang, “The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures”, in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

`muspy.pitch_entropy(music: muspy.music.Music) → float`  
Return the entropy of the normalized note pitch histogram.

The pitch entropy is defined as the Shannon entropy of the normalized note pitch histogram. Drum tracks are ignored. Return NaN if no note is found.

$$\text{pitch\_entropy} = - \sum_{i=0}^{127} P(\text{pitch} = i) \log_2 P(\text{pitch} = i)$$

**Parameters** `music` (`muspy.Music`) – Music object to evaluate.

**Returns** Pitch entropy.

**Return type** `float`

See also:

`muspy.pitch_class_entropy()` Compute the entropy of the normalized pitch class histogram.



`muspy.pitch_in_scale_rate` (*music*: *muspy.music.Music*, *root*: *int*, *mode*: *str*) → float

Return the ratio of pitches in a certain musical scale.

The pitch-in-scale rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$\text{pitch\_in\_scale\_rate} = \frac{\#(\text{notes\_in\_scale})}{\#(\text{notes})}$$

#### Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **root** (*int*) – Root of the scale.
- **mode** (*str*, {'major', 'minor'}) – Mode of the scale.

**Returns** Pitch-in-scale rate.

**Return type** float

See also:

`muspy.scale_consistency()` Compute the largest pitch-in-class rate.

#### References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.pitch_range` (*music*: *muspy.music.Music*) → int

Return the pitch range.

Drum tracks are ignored. Return zero if no note is found.

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Pitch range.

**Return type** int

`muspy.polyphony` (*music*: *muspy.music.Music*) → float

Return the average number of pitches being played concurrently.

The polyphony is defined as the average number of pitches being played at the same time, evaluated only at time steps where at least one pitch is on. Drum tracks are ignored. Return NaN if no note is found.

$$\text{polyphony} = \frac{\#(\text{pitches\_when\_at\_least\_one\_pitch\_is\_on})}{\#(\text{time\_steps\_where\_at\_least\_one\_pitch\_is\_on})}$$

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Polyphony.

**Return type** float

See also:

`muspy.polyphony_rate()` Compute the ratio of time steps where multiple pitches are on.

`muspy.polyphony_rate` (*music*: *muspy.music.Music*, *threshold*: *int* = 2) → float

Return the ratio of time steps where multiple pitches are on.

The polyphony rate is defined as the ratio of the number of time steps where multiple pitches are on to the total number of time steps. Drum tracks are ignored. Return NaN if song length is zero. This metric is used in [1], where it is called *polyphonicity*.

$$\text{polyphony\_rate} = \frac{\#(\text{time\_steps\_where\_multiple\_pitches\_are\_on})}{\#(\text{time\_steps})}$$

#### Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **threshold** (*int*) – Threshold of number of pitches to count into the numerator.

**Returns** Polyphony rate.

**Return type** float

See also:

`muspy.polyphony()` Compute the average number of pitches being played at the same time.

#### References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.scale_consistency` (*music*: *muspy.music.Music*) → float

Return the largest pitch-in-scale rate.

The scale consistency is defined as the largest pitch-in-scale rate over all major and minor scales. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$\text{scale\_consistency} = \max_{\text{root}, \text{mode}} \text{pitch\_in\_scale\_rate}(\text{root}, \text{mode})$$

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Scale consistency.

**Return type** float

See also:

`muspy.pitch_in_scale_rate()` Compute the ratio of pitches in a certain musical scale.

#### References

1. Olof Mogren, “C-RNN-GAN: Continuous recurrent neural networks with adversarial training,” in NeuIPS Workshop on Constructive Machine Learning, 2016.

```
class muspy.Music(metadata: Optional[muspy.classes.Metadata] = None, resolution: Optional[int]
                  = None, tempos: Optional[List[muspy.classes.Tempo]] = None, key_signatures:
                  Optional[List[muspy.classes.KeySignature]] = None, time_signatures: Op-
                  tional[List[muspy.classes.TimeSignature]] = None, downbeats: Optional[List[int]]
                  = None, lyrics: Optional[List[muspy.classes.Lyric]] = None, annota-
                  tions: Optional[List[muspy.classes.Annotation]] = None, tracks: Op-
                  tional[List[muspy.classes.Track]] = None)
```

A universal container for symbolic music.

This is the core class of MusPy. A Music object can be constructed in the following ways.

- `muspy.Music()`: Construct by setting values for attributes.
- `muspy.Music.from_dict()`: Construct from a dictionary that stores the attributes and their values as key-value pairs.
- `muspy.read()`: Read from a MIDI, a MusicXML or an ABC file.
- `muspy.load()`: Load from a JSON or a YAML file saved by `muspy.save()`.
- `muspy.from_object()`: Convert from a `music21.Stream`, a `mido.MidiFile`, a `pretty_midi.PrettyMIDI` or a `pypianoroll.Multitrack` object.

#### **metadata**

Metadata.

**Type** `muspy.Metadata`

#### **resolution**

Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Type** `int`, optional

#### **tempos**

Tempo changes.

**Type** list of `muspy.Tempo`

#### **key\_signatures**

Key signatures changes.

**Type** list of `muspy.KeySignature`

#### **time\_signatures**

Time signature changes.

**Type** list of `muspy.TimeSignature`

#### **downbeats**

Downbeat positions.

**Type** list of `int`

#### **lyrics**

Lyrics.

**Type** list of `muspy.Lyric`

#### **annotations**

Annotations.

**Type** list of `muspy.Annotation`

#### **tracks**

Music tracks.

**Type** list of *muspy.Track*

---

**Note:** Indexing a Music object returns the track of a certain index. That is, `music[idx]` returns `music.tracks[idx]`. Length of a Music object is the number of tracks. That is, `len(music)` returns `len(music.tracks)`.

---

**adjust\_resolution** (*target: Optional[int] = None, factor: Optional[float] = None, rounding: Union[str, Callable, None] = 'round'*) → *muspy.music.Music*  
Adjust resolution and timing of all time-stamped objects.

**Parameters**

- **target** (*int, optional*) – Target resolution.
- **factor** (*int or float, optional*) – Factor used to adjust the resolution based on the formula:  $new\_resolution = old\_resolution * factor$ . For example, a factor of 2 double the resolution, and a factor of 0.5 halve the resolution.
- **rounding** (*{'round', 'ceil', 'floor'} or callable, optional*) – Rounding mode. Defaults to 'round'.

**Returns**

**Return type** Object itself.

**clip** (*lower: int = 0, upper: int = 127*) → *muspy.music.Music*  
Clip the velocity of each note for each track.

**Parameters**

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

**Returns**

**Return type** Object itself.

**get\_end\_time** (*is\_sorted: bool = False*) → *int*  
Return the the time of the last event in all tracks.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations.

**Parameters** **is\_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

**get\_real\_end\_time** (*is\_sorted: bool = False*) → *float*  
Return the end time in realtime.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations. Assume 120 qpm (quarter notes per minute) if no tempo information is available.

**Parameters** **is\_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

**save** (*path: Union[str, pathlib.Path], kind: Optional[str] = None, \*\*kwargs*)  
Save loselessly to a JSON or a YAML file.

Refer to *muspy.save()* for full documentation.

**save\_json** (*path: Union[str, pathlib.Path], \*\*kwargs*)  
Save loselessly to a JSON file.

Refer to *muspy.save\_json()* for full documentation.

**save\_yaml** (*path*: Union[str, pathlib.Path])  
 Save loselessly to a YAML file.  
 Refer to `muspy.save_yaml()` for full documentation.

**show** (*kind*: str, *\*\*kwargs*)  
 Show visualization.  
 Refer to `muspy.show()` for full documentation.

**show\_pianoroll** (*\*\*kwargs*)  
 Show pianoroll visualization.  
 Refer to `muspy.show_pianoroll()` for full documentation.

**show\_score** (*\*\*kwargs*)  
 Show score visualization.  
 Refer to `muspy.show_score()` for full documentation.

**synthesize** (*\*\*kwargs*) → numpy.ndarray  
 Synthesize a Music object to raw audio.  
 Refer to `muspy.synthesize()` for full documentation.

**to\_event\_representation** (*\*\*kwargs*) → numpy.ndarray  
 Return in event-based representation.  
 Refer to `muspy.to_event_representation()` for full documentation.

**to\_mido** (*\*\*kwargs*) → mido.midifiles.midifiles.MidiFile  
 Return as a MidiFile object.  
 Refer to `muspy.to_mido()` for full documentation.

**to\_music21** (*\*\*kwargs*) → music21.stream.Stream  
 Return as a Stream object.  
 Refer to `muspy.to_music21()` for full documentation.

**to\_note\_representation** (*\*\*kwargs*) → numpy.ndarray  
 Return in note-based representation.  
 Refer to `muspy.to_note_representation()` for full documentation.

**to\_object** (*kind*: str, *\*\*kwargs*)  
 Return as an object in other libraries.  
 Refer to `muspy.to_object()` for full documentation.

**to\_pianoroll\_representation** (*\*\*kwargs*) → numpy.ndarray  
 Return in piano-roll representation.  
 Refer to `muspy.to_pianoroll_representation()` for full documentation.

**to\_pitch\_representation** (*\*\*kwargs*) → numpy.ndarray  
 Return in pitch-based representation.  
 Refer to `muspy.to_pitch_representation()` for full documentation.

**to\_pretty\_midi** (*\*\*kwargs*) → pretty\_midi.pretty\_midi.PrettyMIDI  
 Return as a PrettyMIDI object.  
 Refer to `muspy.to_pretty_midi()` for full documentation.

**to\_pypianoroll** (*\*\*kwargs*) → pypianoroll.multitrack.Multitrack  
Return as a Multitrack object.

Refer to `muspy.to_pypianoroll()` for full documentation.

**to\_representation** (*kind: str, \*\*kwargs*) → numpy.ndarray  
Return in a specific representation.

Refer to `muspy.to_representation()` for full documentation.

**transpose** (*semitone: int*) → muspy.music.Music  
Transpose all the notes by a number of semitones.

**Parameters** **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

**Returns**

**Return type** Object itself.

## Notes

Drum tracks are skipped.

**write** (*path: Union[str, pathlib.Path], kind: Optional[str] = None, \*\*kwargs*)  
Write to a MIDI, a MusicXML, an ABC or an audio file.

Refer to `muspy.write()` for full documentation.

**write\_abc** (*path: Union[str, pathlib.Path], \*\*kwargs*)  
Write to an ABC file.

Refer to `muspy.write_abc()` for full documentation.

**write\_audio** (*path: Union[str, pathlib.Path], \*\*kwargs*)  
Write to an audio file.

Refer to `muspy.write_audio()` for full documentation.

**write\_midi** (*path: Union[str, pathlib.Path], \*\*kwargs*)  
Write to a MIDI file.

Refer to `muspy.write_midi()` for full documentation.

**write\_musicxml** (*path: Union[str, pathlib.Path], \*\*kwargs*)  
Write to a MusicXML file.

Refer to `muspy.write_musicxml()` for full documentation.

**muspy.save** (*path: Union[str, pathlib.Path, TextIO], music: Music, kind: Optional[str] = None, \*\*kwargs*)  
Save a Music object loselessly to a JSON or a YAML file.

This is a wrapper function for `muspy.save_json()` and `muspy.save_yaml()`.

## Parameters

- **path** (*str, Path or TextIO*) – Path or file to save the data.
- **music** (*muspy.Music*) – Music object to save.
- **kind** (*{'json', 'yaml'}, optional*) – Format to save. Defaults to infer from the extension.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`.

See also:

**`muspy.save_json()`** Save a Music object to a JSON file.

**`muspy.save_yaml()`** Save a Music object to a YAML file.

**`muspy.write()`** Write a Music object to a MIDI/MusicXML/ABC/audio file.

## Notes

The conversion can be lossy if any nonserializable object is used (for example, an Annotation object, which can store data of any type).

**`muspy.save_json`** (*path*: Union[str, pathlib.Path, TextIO], *music*: Music, *skip\_missing*: bool = True, *ensure\_ascii*: bool = False, *compressed*: Optional[bool] = None, *\*\*kwargs*)  
Save a Music object to a JSON file.

### Parameters

- **`path`** (*str*, *Path* or *TextIO*) – Path or file to save the JSON data.
- **`music`** (*muspy.Music*) – Music object to save.
- **`skip_missing`** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **`ensure_ascii`** (*bool*) – Whether to escape non-ASCII characters. Will be passed to PyYAML’s *yaml.dump*. Defaults to False.
- **`compressed`** (*bool*, *optional*) – Whether to save as a compressed JSON file (*.json.gz*). Has no effect when *path* is a file object. Defaults to infer from the extension (*.gz*).
- **`**kwargs`** – Keyword arguments to pass to *json.dumps()*.

## Notes

When a path is given, use UTF-8 encoding and gzip compression if *compressed=True*.

**`muspy.save_yaml`** (*path*: Union[str, pathlib.Path, TextIO], *music*: Music, *skip\_missing*: bool = True, *allow\_unicode*: bool = True, *compressed*: Optional[bool] = None, *\*\*kwargs*)  
Save a Music object to a YAML file.

### Parameters

- **`path`** (*str*, *Path* or *TextIO*) – Path or file to save the YAML data.
- **`music`** (*muspy.Music*) – Music object to save.
- **`skip_missing`** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **`allow_unicode`** (*bool*) – Whether to escape non-ASCII characters. Will be passed to *json.dumps()*. Defaults to False.
- **`compressed`** (*bool*, *optional*) – Whether to save as a compressed YAML file (*.yaml.gz*). Has no effect when *path* is a file object. Defaults to infer from the extension (*.gz*).
- **`**kwargs`** – Keyword arguments to pass to *yaml.dump*.

## Notes

When a path is given, use UTF-8 encoding and gzip compression if *compressed=True*.

```
muspy.to_event_representation(music: Music, use_single_note_off_event: bool = False,
                              use_end_of_sequence_event: bool = False, encode_velocity: bool
                              = False, force_velocity_event: bool = True, max_time_shift: int =
                              100, velocity_bins: int = 32) → numpy.ndarray
```

Encode a Music object into event-based representation.

The event-based representation represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is  $M \times 1$ , where  $M$  is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-on events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

### Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use\_single\_note\_off\_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use\_end\_of\_sequence\_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode velocities.
- **force\_velocity\_event** (*bool*) – Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.
- **max\_time\_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max\_time\_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity\_bins** (*int*) – Number of velocity bins to use. Defaults to 32.

**Returns** Encoded array in event-based representation.

**Return type** ndarray, dtype=uint16, shape=(?, 1)

```
muspy.to_mido(music: Music, use_note_off_message: bool = False)
```

Return a Music object as a MidiFile object.

### Parameters

- **music** (*muspy.Music* object) – Music object to convert.
- **use\_note\_off\_message** (*bool*, *optional*) – Whether to use note-off messages. If False, note-on messages with zero velocity are used instead. The advantage to using note-on messages at zero velocity is that it can avoid sending additional status bytes when Running Status is employed. Defaults to False.

**Returns** Converted MidiFile object.

**Return type** *mido.MidiFile*

```
muspy.to_music21(music: Music) → music21.stream.Score
```

Return a Music object as a music21 Score object.

**Parameters** **music** (*muspy.Music*) – Music object to convert.

**Returns** Converted music21 Score object.



**Return type** *music21.stream.Score*

`muspy.to_note_representation` (*music*: *Music*, *use\_start\_end*: *bool* = *False*, *encode\_velocity*: *bool* = *True*, *dtype*: *Union[numpy.dtype, type, str]* = *<class 'int'>*) → *numpy.ndarray*

Encode a Music object into note-based representation.

The note-based representation represents music as a sequence of (time, pitch, duration, velocity) tuples. For example, a note `Note(time=0, duration=4, pitch=60, velocity=64)` will be encoded as a tuple (0, 60, 4, 64). The output shape is *N \* D*, where *N* is the number of notes and *D* is 4 when *encode\_velocity* is *True*, otherwise *D* is 3. The values of the second dimension represent time, pitch, duration and velocity (discarded when *encode\_velocity* is *False*).

#### Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use\_start\_end** (*bool*) – Whether to use ‘start’ and ‘end’ to encode the timing rather than ‘time’ and ‘duration’. Defaults to *False*.
- **encode\_velocity** (*bool*) – Whether to encode note velocities. Defaults to *True*.
- **dtype** (*dtype, type or str*) – Data type of the return array. Defaults to *int*.

**Returns** Encoded array in note-based representation.

**Return type** *ndarray*, shape=(?, 3 or 4)

`muspy.to_object` (*music*: *Music*, *kind*: *str*, *\*\*kwargs*) → *Union[music21.stream.Stream, mido.midifiles.MidiFile, pretty\_midi.pretty\_midi.PrettyMIDI, pypianoroll.multitrack.Multitrack]*

Return a Music object as an object in other libraries.

Supported classes are *music21.Stream*, *mido.MidiTrack*, *pretty\_midi.PrettyMIDI* and *pypianoroll.Multitrack*.

#### Parameters

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str*, {'music21', 'mido', 'pretty\_midi', 'pypianoroll'}) – Target class.

**Returns** Converted object.

**Return type** *music21.Stream*, *mido.MidiTrack*, *pretty\_midi.PrettyMIDI* or *pypianoroll.Multitrack*

`muspy.to_pianoroll_representation` (*music*: *Music*, *encode\_velocity*: *bool* = *True*) → *numpy.ndarray*

Encode notes into piano-roll representation.

#### Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **encode\_velocity** (*bool*) – Whether to encode velocities. If *True*, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to *True*.

**Returns** Encoded array in piano-roll representation.

**Return type** *ndarray*, dtype=*uint8* or *bool*, shape=(?, 128)

`muspy.to_pitch_representation` (*music*: *Music*, *use\_hold\_state*: *bool* = *False*) → *numpy.ndarray*

Encode a Music object into pitch-based representation.

The pitch-based representation represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is  $T \times 1$ , where  $T$  is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

**Parameters**

- **music** (*muspy.Music*) – Music object to encode.
- **use\_hold\_state** (*bool*) – Whether to use a special state for holds. Defaults to False.

**Returns** Encoded array in pitch-based representation.

**Return type** ndarray, dtype=uint8, shape=(?, 1)

`muspy.to_pretty_midi (music: Music) → pretty_midi.pretty_midi.PrettyMIDI`

Return a Music object as a PrettyMIDI object.

Tempo changes are not supported yet.

**Parameters** **music** (*muspy.Music* object) – Music object to convert.

**Returns** Converted PrettyMIDI object.

**Return type** `pretty_midi.PrettyMIDI`

**Notes**

Tempo information will not be included in the output.

`muspy.to_pypianoroll (music: Music) → pypianoroll.multitrack.Multitrack`

Return a Music object as a Multitrack object.

**Parameters** **music** (*muspy.Music*) – Music object to convert.

**Returns** **multitrack** – Converted Multitrack object.

**Return type** `pypianoroll.Multitrack`

`muspy.to_representation (music: Music, kind: str, **kwargs) → numpy.ndarray`

Return a Music object in a specific representation.

**Parameters**

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str*, {'pitch', 'piano-roll', 'event', 'note'}) – Target representation.

**Returns** **array** – Converted representation.

**Return type** ndarray

`muspy.synthesize (music: Music, soundfont_path: Union[str, pathlib.Path, None] = None, rate: int = 44100) → numpy.ndarray`

Synthesize a Music object to raw audio.

**Parameters**

- **music** (*muspy.Music*) – Music object to write.
- **soundfont\_path** (*str* or *Path*, optional) – Path to the soundfont file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.

**Returns** Synthesized waveform.

**Return type** ndarray, dtype=int16, shape=(?, 2)

`muspy.write(path: Union[str, pathlib.Path], music: Music, kind: Optional[str] = None, **kwargs)`  
Write a Music object to a MIDI/MusicXML/ABC/audio file.

#### Parameters

- **path** (*str* or *Path*) – Path to write the file.
- **music** (*muspy.Music*) – Music object to convert.
- **kind** ({'midi', 'musicxml', 'abc', 'audio'}, *optional*) – Format to save. Defaults to infer from the extension.

See also:

`muspy.save()` Save a Music object loselessly to a JSON or a YAML file.

`muspy.write_abc(path: Union[str, pathlib.Path], music: Music)`  
Write a Music object to a ABC file.

#### Parameters

- **path** (*str* or *Path*) – Path to write the ABC file.
- **music** (*muspy.Music*) – Music object to write.

`muspy.write_audio(path: Union[str, pathlib.Path], music: Music, soundfont_path: Union[str, pathlib.Path, None] = None, rate: int = 44100, audio_format: Optional[str] = None)`  
Write a Music object to an audio file.

Supported formats include WAV, AIFF, FLAC and OGA.

#### Parameters

- **path** (*str* or *Path*) – Path to write the audio file.
- **music** (*muspy.Music*) – Music object to write.
- **soundfont\_path** (*str* or *Path*, *optional*) – Path to the soundfont file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.
- **audio\_format** (*str*, {'wav', 'aiff', 'flac', 'oga'}, *optional*) – File format to write. If None, infer it from the extension.

`muspy.write_midi(path: Union[str, pathlib.Path], music: Music, backend: str = 'mido', **kwargs)`  
Write a Music object to a MIDI file.

#### Parameters

- **path** (*str* or *Path*) – Path to write the MIDI file.
- **music** (*muspy.Music*) – Music object to write.
- **backend** ({'mido', 'pretty\_midi'}, *optional*) – Backend to use. Defaults to 'mido'.

See also:

`write_midi_mido()` Write a Music object to a MIDI file using mido as backend.

`write_midi_pretty_midi()` Write a Music object to a MIDI file using pretty\_midi as backend.

`muspy.write_musicxml` (*path*: Union[str, pathlib.Path], *music*: Music, *compressed*: Optional[bool] = None)

Write a Music object to a MusicXML file.

**Parameters**

- **path** (*str* or *Path*) – Path to write the MusicXML file.
- **music** (*muspy.Music*) – Music object to write.
- **compressed** (*bool*, *optional*) – Whether to write to a compressed MusicXML file. If None, infer from the extension of the filename (‘.xml’ and ‘.musicxml’ for an uncompressed file, ‘.mxl’ for a compressed file).

**class** `muspy.NoteRepresentationProcessor` (*use\_start\_end*: bool = False, *encode\_velocity*: bool = True, *dtype*: Union[numpy.dtype, type, str] = <class ‘int’>, *default\_velocity*: int = 64)

Note-based representation processor.

The note-based representation represents music as a sequence of (pitch, time, duration, velocity) tuples. For example, a note `Note(time=0, duration=4, pitch=60, velocity=64)` will be encoded as a tuple (0, 4, 60, 64). The output shape is `L * D`, where `L` is the number of notes and `D` is 4 when `encode_velocity` is True, otherwise `D` is 3. The values of the second dimension represent pitch, time, duration and velocity (discarded when `encode_velocity` is False).

**use\_start\_end**

Whether to use ‘start’ and ‘end’ to encode the timing rather than ‘time’ and ‘duration’. Defaults to False.

**Type** bool

**encode\_velocity**

Whether to encode note velocities. Defaults to True.

**Type** bool

**dtype**

Data type of the return array. Defaults to int.

**Type** dtype, type or str

**default\_velocity**

Default velocity value to use when decoding if `encode_velocity` is False. Defaults to 64.

**Type** int

**decode** (*array*: numpy.ndarray) → `muspy.music.Music`

Decode note-based representation into a Music object.

**Parameters** **array** (*ndarray*) – Array in note-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** `muspy.Music` object

**See also:**

`muspy.from_note_representation()` Return a Music object converted from note-based representation.

**encode** (*music*: `muspy.music.Music`) → numpy.ndarray

Encode a Music object into note-based representation.

**Parameters** **music** (`muspy.Music` object) – Music object to encode.

**Returns** Encoded array in note-based representation.

**Return type** ndarray (np.uint8)

**See also:**

`muspy.to_note_representation()` Convert a Music object into note-based representation.

```
class muspy.EventRepresentationProcessor (use_single_note_off_event: bool = False,
                                         use_end_of_sequence_event: bool =
                                         False, encode_velocity: bool = False,
                                         force_velocity_event: bool = True,
                                         max_time_shift: int = 100, velocity_bins:
                                         int = 32, default_velocity: int = 64)
```

Event-based representation processor.

The event-based representation represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is  $M \times 1$ , where  $M$  is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-on events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

**use\_single\_note\_off\_event**

Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.

**Type** bool

**use\_end\_of\_sequence\_event**

Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.

**Type** bool

**encode\_velocity**

Whether to encode velocities.

**Type** bool

**force\_velocity\_event**

Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.

**Type** bool

**max\_time\_shift**

Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max\_time\_shift* will be decomposed into two or more time-shift events. Defaults to 100.

**Type** int

**velocity\_bins**

Number of velocity bins to use. Defaults to 32.

**Type** int

**default\_velocity**

Default velocity value to use when decoding. Defaults to 64.

**Type** int

**decode** (array: numpy.ndarray) → muspy.music.Music

Decode event-based representation into a Music object.

**Parameters** **array** (ndarray) – Array in event-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** `muspy.Music` object

**See also:**

`muspy.from_event_representation()` Return a Music object converted from event-based representation.

**encode** (*music: muspy.music.Music*) → `numpy.ndarray`  
Encode a Music object into event-based representation.

**Parameters** **music** (`muspy.Music` object) – Music object to encode.

**Returns** Encoded array in event-based representation.

**Return type** `ndarray (np.uint16)`

**See also:**

`muspy.to_event_representation()` Convert a Music object into event-based representation.

**class** `muspy.PianoRollRepresentationProcessor` (*encode\_velocity: bool = True, default\_velocity: int = 64*)

Piano-roll representation processor.

The piano-roll representation represents music as a time-pitch matrix, where the columns are the time steps and the rows are the pitches. The values indicate the presence of pitches at different time steps. The output shape is `T x 128`, where `T` is the number of time steps.

**encode\_velocity**

Whether to encode velocities. If `True`, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to `True`.

**Type** `bool`

**default\_velocity**

Default velocity value to use when decoding if `encode_velocity` is `False`. Defaults to `64`.

**Type** `int`

**decode** (*array: numpy.ndarray*) → `muspy.music.Music`  
Decode piano-roll representation into a Music object.

**Parameters** **array** (`ndarray`) – Array in piano-roll representation to decode. Cast to integer if not of integer type. If `encode_velocity` is `True`, casted to boolean if not of boolean type.

**Returns** Decoded Music object.

**Return type** `muspy.Music` object

**See also:**

`muspy.from_pianoroll_representation()` Return a Music object converted from piano-roll representation.

**encode** (*music: muspy.music.Music*) → `numpy.ndarray`  
Encode a Music object into piano-roll representation.

**Parameters** **music** (`muspy.Music` object) – Music object to encode.

**Returns** Encoded array in piano-roll representation.

**Return type** `ndarray (np.uint8)`

See also:

`muspy.to_pianoroll_representation()` Convert a Music object into piano-roll representation.

**class** `muspy.PitchRepresentationProcessor` (*use\_hold\_state: bool = False, default\_velocity: int = 64*)

Pitch-based representation processor.

The pitch-based representation represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is  $T \times 1$ , where  $T$  is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

**use\_hold\_state**

Whether to use a special state for holds. Defaults to False.

**Type** `bool`

**default\_velocity**

Default velocity value to use when decoding. Defaults to 64.

**Type** `int`

**decode** (*array: numpy.ndarray*)  $\rightarrow$  `muspy.music.Music`

Decode pitch-based representation into a Music object.

**Parameters** **array** (*ndarray*) – Array in pitch-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** `muspy.Music` object

See also:

`muspy.from_pitch_representation()` Return a Music object converted from pitch-based representation.

**encode** (*music: muspy.music.Music*)  $\rightarrow$  `numpy.ndarray`

Encode a Music object into pitch-based representation.

**Parameters** **music** (*muspy.Music* object) – Music object to encode.

**Returns** Encoded array in pitch-based representation.

**Return type** `ndarray (np.uint8)`

See also:

`muspy.to_pitch_representation()` Convert a Music object into pitch-based representation.

`muspy.get_json_schema_path()`  $\rightarrow$  `str`

Return the path to the JSON schema.

`muspy.get_musicxml_schema_path()`  $\rightarrow$  `str`

Return the path to the MusicXML schema.

`muspy.get_yaml_schema_path()`  $\rightarrow$  `str`

Return the path to the YAML schema.

`muspy.validate_json` (*path: Union[str, pathlib.Path]*)

Validate a file against the JSON schema.

**Parameters** `path` (*str* or *Path*) – Path to the file to validate.

`muspy.validate_musicxml` (*path*: *Union[str, pathlib.Path]*)

Validate a file against the MusicXML schema.

**Parameters** `path` (*str* or *Path*) – Path to the file to validate.

`muspy.validate_yaml` (*path*: *Union[str, pathlib.Path]*)

Validate a file against the YAML schema.

**Parameters** `path` (*str* or *Path*) – Path to the file to validate.

`muspy.show` (*music*: *Music*, *kind*: *str*, *\*\*kwargs*)

Show visualization.

**Parameters**

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str*, {*'piano-roll'*, *'score'*}) – Target representation.

`muspy.show_pianoroll` (*music*: *Music*, *\*\*kwargs*)

Show pianoroll visualization.

`muspy.show_score` (*music*: *Music*, *figsize*: *Optional[Tuple[float, float]]* = *None*, *clef*: *str* = *'treble'*, *clef\_octave*: *Optional[int]* = *0*, *note\_spacing*: *Optional[int]* = *None*, *font\_path*: *Union[str, pathlib.Path, None]* = *None*, *font\_scale*: *Optional[float]* = *None*) → *muspy.visualization.score.ScorePlotter*

Show score visualization.

**Parameters**

- **music** (*muspy.Music*) – Music object to show.
- **figsize** ((*float*, *float*), *optional*) – Width and height in inches. Defaults to Matplotlib configuration.
- **clef** (*str*, {*'treble'*, *'alto'*, *'bass'*}) – Clef type. Defaults to a treble clef.
- **clef\_octave** (*int*) – Clef octave. Defaults to zero.
- **note\_spacing** (*int*, *optional*) – Spacing of notes. Defaults to 4.
- **font\_path** (*str* or *Path*, *optional*) – Path to the music font. Defaults to the path to the built-in Bravura font.
- **font\_scale** (*float*, *optional*) – Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

**Returns** A *ScorePlotter* object that handles the score.

**Return type** *muspy.ScorePlotter*

**class** `muspy.ScorePlotter` (*fig*: *matplotlib.figure.Figure*, *ax*: *matplotlib.axes.\_axes.Axes*, *resolution*: *int*, *note\_spacing*: *Optional[int]* = *None*, *font\_path*: *Union[str, pathlib.Path, None]* = *None*, *font\_scale*: *Optional[float]* = *None*)

A plotter that handles the score visualization.

**fig**

Figure object to plot the score on.

**Type** *matplotlib.figure.Figure*

**axes**

Axes object to plot the score on.

**Type** *matplotlib.axes.Axes*



**resolution**  
Time steps per quarter note.  
**Type** `int`

**note\_spacing**  
Spacing of notes. Defaults to 4.  
**Type** `int`, optional

**font\_path**  
Path to the music font. Defaults to the path to the downloaded Bravura font.  
**Type** `str` or `Path`, optional

**font\_scale**  
Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.  
**Type** `float`, optional

**adjust\_fonts** (*scale: Optional[float] = None*)  
Adjust the fonts.

**plot\_bar\_line** () → `matplotlib.lines.Line2D`  
Plot a bar line.

**plot\_clef** (*kind='treble', octave=0*) → `matplotlib.text.Text`  
Plot a clef.

**plot\_final\_bar\_line** () → `List[matplotlib.artist.Artist]`  
Plot an ending bar line.

**plot\_key\_signature** (*root: int, mode: str*)  
Plot a key signature. Supports only major and minor keys.

**plot\_note** (*time, duration, pitch*) → `Optional[Tuple[List[matplotlib.text.Text], List[matplotlib.patches.Arc]]]`  
Plot a note.

**plot\_object** (*obj*)  
Plot an object.

**plot\_staffs** (*start: Optional[float] = None, end: Optional[float] = None*) → `List[matplotlib.lines.Line2D]`  
Plot the staffs.

**plot\_tempo** (*qpm*) → `List[matplotlib.artist.Artist]`  
Plot a tempo as a metronome mark.

**plot\_time\_signature** (*numerator: int, denominator: int*) → `List[matplotlib.text.Text]`  
Plot a time signature.

**set\_baseline** (*y*)  
Set baseline position (y-coordinate of first staff line).

**update\_boundaries** (*left: Optional[float] = None, right: Optional[float] = None, bottom: Optional[float] = None, top: Optional[float] = None*)  
Update boundaries.

## 7.10.2 muspy.datasets

Dataset classes.

This module provides an easy-to-use dataset management system. Each supported dataset in MusPy comes with a class inherited from the base MusPy Dataset class. It also provides interfaces to PyTorch and TensorFlow for creating input pipelines for machine learning.

## Base Classes

- ABCFolderDataset
- Dataset
- DatasetInfo
- FolderDataset
- RemoteABCFolderDataset
- RemoteDataset
- RemoteFolderDataset
- RemoteMusicDataset
- MusicDataset

## Dataset Classes

- EssenFolkSongDatabase
- HaydnOp20Dataset
- HymnalDataset
- HymnalTuneDataset
- JSBChoralesDataset
- LakhMIDIAliignedDataset
- LakhMIDIDataset
- LakhMIDIMatchedDataset
- MAESTRODatasetV1
- MAESTRODatasetV2
- Music21Dataset
- MusicNetDataset
- NESMusicDatabase
- NottinghamDatabase
- WikifoniaDataset

```
class muspy.datasets.ABCFolderDataset (root: Union[str, pathlib.Path], convert: Optional[bool] = None, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None, use_converted: Optional[bool] = None)
```

Class for datasets storing ABC files in a folder.

**See also:**

[\*\*\*muspy.FolderDataset\*\*\*](#) Class for datasets storing files in a folder.

**on\_the\_fly**( ) → FolderDatasetType  
Enable on-the-fly mode and convert the data on the fly.

### Returns

**Return type** Object itself.

**read**(filename: Tuple[str, Tuple[int, int]]) → muspy.music.Music  
Read a file into a Music object.

**class** muspy.datasets.Dataset  
Base class for MusPy datasets.

To build a custom dataset, it should inherit this class and override the methods `__getitem__` and `__len__` as well as the class attribute `_info`. `__getitem__` should return the *i*-th data sample as a *muspy.Music* object. `__len__` should return the size of the dataset. `_info` should be a *muspy.DatasetInfo* instance storing the dataset information.

**classmethod** citation()  
Print the citation information.

**classmethod** info()  
Return the dataset information.

**save**(root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n\_jobs: int = 1, ignore\_exceptions: bool = True, verbose: bool = True, \*\*kwargs)  
Save all the music objects to a directory.

### Parameters

- **root** (*str* or *Path*) – Root directory to save the data.
- **kind** ({'json', 'yaml'}, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.
- **\*\*kwargs** – Keyword arguments to pass to *muspy.save\_json()* or *muspy.save\_yaml()*

### Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

**split**(filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None) → Dict[str, List[int]]  
Return the dataset as a PyTorch dataset.

### Parameters

- **filename** (*str* or *Path*, *optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float* or *list of float*, *optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and

test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If *int* or *array\_like*, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**to\_pytorch\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If *None* or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If *None*, return the full dataset as a whole. If *float*, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random\_state** (*int, array\_like or RandomState, optional*) – Random state used to create the splits. If *int* or *array\_like*, the value is passed to `numpy.random.RandomState`, and the created `RandomState` object is used to create the splits. If `RandomState`, it will be used to create the splits.

**Returns** Converted PyTorch dataset(s).

**Return type** `class:torch.utils.data.Dataset` or `Dict of :class:torch.utils.data.Dataset`

**to\_tensorflow\_dataset** (*factory: Optional[Callable] = None, representation: Optional[str] = None, split\_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random\_state: Any = None, \*\*kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

#### Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See `muspy.to_representation()` for available representation.
- **split\_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If *None* or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If *None*, return the full dataset as a whole. If *float*, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random\_state** (*int*, *array\_like* or *RandomState*, *optional*) – Random state used to create the splits. If *int* or *array\_like*, the value is passed to `numpy.random.RandomState`, and the created *RandomState* object is used to create the splits. If *RandomState*, it will be used to create the splits.

### Returns

- `class:tensorflow.data.Dataset` or Dict of
- `class:tensorflow.data.dataset` – Converted TensorFlow dataset(s).

**class** `muspy.datasets.DatasetInfo` (*name: Optional[str] = None*, *description: Optional[str] = None*, *homepage: Optional[str] = None*, *license: Optional[str] = None*)

A container for dataset information.

**class** `muspy.datasets.EssenFolkSongDatabase` (*root: Union[str, pathlib.Path]*, *download\_and\_extract: bool = False*, *overwrite: Optional[bool] = None*, *cleanup: Optional[bool] = None*, *convert: Optional[bool] = None*, *kind: str = 'json'*, *n\_jobs: int = 1*, *ignore\_exceptions: Optional[bool] = None*, *use\_converted: Optional[bool] = None*, *verbose: bool = True*)

Essen Folk Song Database.

**class** `muspy.datasets.FolderDataset` (*root: Union[str, pathlib.Path]*, *convert: Optional[bool] = None*, *kind: str = 'json'*, *n\_jobs: int = 1*, *ignore\_exceptions: Optional[bool] = None*, *use\_converted: Optional[bool] = None*)

Class for datasets storing files in a folder.

This class extends `muspy.Dataset` to support folder datasets. To build a custom folder dataset, please refer to the documentation of `muspy.Dataset` for details. In addition, set class attribute `_extension` to the extension to look for when building the dataset and set `read` to a callable that takes as inputs a filename of a source file and return the converted Music object.

### root

Root directory of the dataset.

Type `str` or `Path`

### Parameters

- **convert** (*bool*, *optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If *False*, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to *False*.
- **kind** (*{'json', 'yaml'}*, *optional*) – File format to save the data. Defaults to *'json'*.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to *True*.
- **use\_converted** (*bool*, *optional*) – Force to disable on-the-fly mode and use stored converted data

---

**Important:** `muspy.FolderDataset.converted_exists()` depends solely on a special file named `.muspy.success` in the folder `{root}/_converted/`, which serves as an indicator for the existence and integrity of the converted dataset. If the converted dataset is built by `muspy.FolderDataset.convert()`, the `.muspy.success` file will be created as well. If the converted dataset is created manually, make sure to create the `.muspy.success` file in the folder `{root}/_converted/` to prevent errors.

---

## Notes

Two modes are available for this dataset. When the on-the-fly mode is enabled, a data sample is converted to a music object on the fly when being indexed. When the on-the-fly mode is disabled, a data sample is loaded from the precomputed converted data.

See also:

**`muspy.Dataset`** Base class for MusPy datasets.

**`convert`** (*kind*: str = 'json', *n\_jobs*: int = 1, *ignore\_exceptions*: bool = True, *verbose*: bool = True) → FolderDatasetType  
Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

### Parameters

- **`kind`** ({'json', 'yaml'}, optional) – File format to save the data. Defaults to 'json'.
- **`n_jobs`** (int, optional) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **`ignore_exceptions`** (bool, optional) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **`verbose`** (bool, optional) – Whether to be verbose. Defaults to True.

### Returns

**Return type** Object itself.

**`converted_dir`**  
Path to the root directory of the converted dataset.

**`converted_exists`** () → bool  
Return True if the saved dataset exists, otherwise False.

**`exists`** () → bool  
Return True if the dataset exists, otherwise False.

**`load`** (*filename*: Union[str, pathlib.Path]) → muspy.music.Music  
Load a file into a Music object.

**`on_the_fly`** () → FolderDatasetType  
Enable on-the-fly mode and convert the data on the fly.

### Returns

**Return type** Object itself.

**read** (*filename: Any*) → muspy.music.Music  
Read a file into a Music object.

**use\_converted** () → FolderDatasetType  
Disable on-the-fly mode and use converted data.

**Returns**

**Return type** Object itself.

```
class muspy.datasets.HaydnOp20Dataset (root: Union[str, pathlib.Path], download_and_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None, use_converted: Optional[bool] = None, verbose: bool = True)
```

Haydn Op.20 Dataset.

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Read a file into a Music object.

```
class muspy.datasets.HymnalDataset (root: Union[str, pathlib.Path], download: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None)
```

Hymnal Dataset.

**download** () → muspy.datasets.base.FolderDataset  
Download the source datasets.

**Returns**

**Return type** Object itself.

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Read a file into a Music object.

```
class muspy.datasets.HymnalTuneDataset (root: Union[str, pathlib.Path], download: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None)
```

Hymnal Dataset (tune only).

**download** () → muspy.datasets.base.FolderDataset  
Download the source datasets.

**Returns**

**Return type** Object itself.

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Read a file into a Music object.

```
class muspy.datasets.JSBChoralesDataset (root: Union[str, pathlib.Path], download_and_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None, use_converted: Optional[bool] = None, verbose: bool = True)
```

Johann Sebastian Bach Chorales Dataset.

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Read a file into a Music object.

```
class muspy.datasets.LakhMIDIAlignedDataset (root: Union[str, pathlib.Path], download_and_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None, use_converted: Optional[bool] = None, verbose: bool = True)
```

Lakh MIDI Dataset - aligned subset.

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Read a file into a Music object.

```
class muspy.datasets.LakhMIDIDataset (root: Union[str, pathlib.Path], download_and_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None, use_converted: Optional[bool] = None, verbose: bool = True)
```

Lakh MIDI Dataset.

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Read a file into a Music object.

```
class muspy.datasets.LakhMIDIMatchedDataset (root: Union[str, pathlib.Path], download_and_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None, use_converted: Optional[bool] = None, verbose: bool = True)
```

Lakh MIDI Dataset - matched subset.

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Read a file into a Music object.

```
class muspy.datasets.MAESTRODatasetV1 (root: Union[str, pathlib.Path], download_and_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None, use_converted: Optional[bool] = None, verbose: bool = True)
```

MAESTRO Dataset (MIDI only).

**read** (*filename: Union[str, pathlib.Path]*) → muspy.music.Music  
Read a file into a Music object.



```
class muspy.datasets.MAESTRODatasetV2 (root: Union[str, pathlib.Path],
                                         download_and_extract: bool = False,
                                         overwrite: Optional[bool] = None,
                                         cleanup: Optional[bool] = None,
                                         convert: Optional[bool] = None,
                                         kind: str = 'json',
                                         n_jobs: int = 1,
                                         ignore_exceptions: Optional[bool] = None,
                                         use_converted: Optional[bool] = None,
                                         verbose: bool = True)
```

MAESTRO Dataset (MIDI only).

**read** (filename: Union[str, pathlib.Path]) → muspy.music.Music  
Read a file into a Music object.

```
class muspy.datasets.Music21Dataset (composer: Optional[str] = None)
A class of datasets containing files in music21 corpus.
```

#### Parameters

- **composer** (*str*) – Name of a composer or a collection. Please refer to the music21 corpus reference page for a full list [1].
- **extensions** (*list of str*) – File extensions of desired files.

#### References

[1] <https://web.mit.edu/music21/doc/about/referenceCorpus.html>

```
convert (root: Union[str, pathlib.Path], kind: str = 'json', n_jobs: int = 1,
         ignore_exceptions: bool = True) → muspy.datasets.base.MusicDataset
Convert and save the Music objects.
```

#### Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** ({'json', 'yaml'}, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

```
class muspy.datasets.MusicDataset (root: Union[str, pathlib.Path], kind: str = 'json')
Class for datasets of MusPy JSON/YAML files.
```

**root**  
Root directory of the dataset.

**Type** *str* or *Path*

**kind**  
File format of the data. Defaults to 'json'.

**Type** {'json', 'yaml'}, *optional*

See also:

*[muspy.Dataset](#)* Base class for MusPy datasets.

```
class muspy.datasets.MusicNetDataset (root: Union[str, pathlib.Path], download_and_extract:
    bool = False, overwrite: Optional[bool] = None,
    cleanup: Optional[bool] = None, convert: Op-
    tional[bool] = None, kind: str = 'json', n_jobs:
    int = 1, ignore_exceptions: Optional[bool] = None,
    use_converted: Optional[bool] = None, verbose: bool
    = True)
```

MusicNet Dataset (MIDI only).

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.datasets.NESMusicDatabase (root: Union[str, pathlib.Path], down-
    load_and_extract: bool = False, overwrite: Op-
    tional[bool] = None, cleanup: Optional[bool] =
    None, convert: Optional[bool] = None, kind: str
    = 'json', n_jobs: int = 1, ignore_exceptions: Op-
    tional[bool] = None, use_converted: Optional[bool]
    = None, verbose: bool = True)
```

NES Music Database.

```
read (filename: Union[str, pathlib.Path]) → muspy.music.Music
    Read a file into a Music object.
```

```
class muspy.datasets.NottinghamDatabase (root: Union[str, pathlib.Path], down-
    load_and_extract: bool = False, overwrite:
    Optional[bool] = None, cleanup: Optional[bool]
    = None, convert: Optional[bool] = None, kind:
    str = 'json', n_jobs: int = 1, ignore_exceptions:
    Optional[bool] = None, use_converted: Op-
    tional[bool] = None, verbose: bool = True)
```

Nottingham Database.

```
class muspy.datasets.RemoteABCFolderDataset (root: Union[str, pathlib.Path], down-
    load_and_extract: bool = False, over-
    write: Optional[bool] = None, cleanup:
    Optional[bool] = None, convert: Op-
    tional[bool] = None, kind: str = 'json',
    n_jobs: int = 1, ignore_exceptions: Op-
    tional[bool] = None, use_converted:
    Optional[bool] = None, verbose: bool =
    True)
```

Base class for remote datasets storing ABC files in a folder.

See also:

***muspy.ABCFolderDataset*** Class for datasets storing ABC files in a folder.

***muspy.RemoteDataset*** Base class for remote MusPy datasets.

```
class muspy.datasets.RemoteDataset (root: Union[str, pathlib.Path], download_and_extract:
    Optional[bool] = None, overwrite: Optional[bool] =
    None, cleanup: Optional[bool] = None, verbose: bool =
    True)
```

Base class for remote MusPy datasets.

This class extends *muspy.Dataset* to support remote datasets. To build a custom remote dataset, please refer to the documentation of *muspy.Dataset* for details. In addition, set the class attribute `_sources` to the URLs to the source files (see Notes).

**root**

Root directory of the dataset.

Type `str` or `Path`

**Parameters**

- **download\_and\_extract** (*bool*, *optional*) – Whether to download and extract the dataset. Defaults to `False`.
- **overwrite** (*bool*, *optional*) – Whether to overwrite existing downloaded files. Defaults to `True`.
- **cleanup** (*bool*, *optional*) – Whether to remove the source archive(s). Defaults to `False`.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to `True`.

**Raises** `RuntimeError`: – If `download_and_extract` is `False` but file `{root}/.muspy.` success does not exist (see below).

---

**Important:** `muspy.Dataset.exists()` depends solely on a special file named `.muspy.success` in directory `{root}/_converted/`. This file serves as an indicator for the existence and integrity of the dataset. It will automatically be created if the dataset is successfully downloaded and extracted by `muspy.Dataset.download_and_extract()`. If the dataset is downloaded manually, make sure to create the `.muspy.success` file in directory `{root}/_converted/` to prevent errors.

---

**Notes**

The class attribute `_sources` is a dictionary storing the following information of each source file.

- `filename` (`str`): Name to save the file.
- `url` (`str`): URL to the file.
- `archive` (`bool`): Whether the file is an archive.
- `md5` (`str`, *optional*): Expected MD5 checksum of the file.

Here is an example.:

```
_sources = {
    "example": {
        "filename": "example.tar.gz",
        "url": "https://www.example.com/example.tar.gz",
        "archive": True,
        "md5": None,
    }
}
```

See also:

**`muspy.Dataset`** Base class for MusPy datasets.

**`download`** (*overwrite: bool = True, verbose: bool = True*) → `RemoteDatasetType`

Download the source datasets.

**Parameters**

- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files. Defaults to True.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**download\_and\_extract** (*overwrite: bool = True, cleanup: bool = False, verbose: bool = True*) → RemoteDatasetType  
Download source datasets and extract the downloaded archives.

**Parameters**

- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files. Defaults to True.
- **cleanup** (*bool*, *optional*) – Whether to remove the source archive. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**exists** () → bool  
Return True if the dataset exists, otherwise False.

**extract** (*cleanup: bool = False, verbose: bool = True*) → RemoteDatasetType  
Extract the downloaded archive(s).

**Parameters**

- **cleanup** (*bool*, *optional*) – Whether to remove the source archive after extraction. Defaults to False.
- **verbose** (*bool*, *optional*) – Whether to be verbose. Defaults to True.

**Returns**

**Return type** Object itself.

**source\_exists** () → bool  
Return True if all the sources exist, otherwise False.

**class** muspy.datasets.RemoteFolderDataset (*root: Union[str, pathlib.Path], download\_and\_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n\_jobs: int = 1, ignore\_exceptions: Optional[bool] = None, use\_converted: Optional[bool] = None, verbose: bool = True*)

Base class for remote datasets storing files in a folder.

**root**  
Root directory of the dataset.

**Type** str or Path

**Parameters**

- **download\_and\_extract** (*bool*, *optional*) – Whether to download and extract the dataset. Defaults to False.

- **cleanup** (*bool*, *optional*) – Whether to remove the source archive(s). Defaults to False.
- **convert** (*bool*, *optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** ({'json', 'yaml'}, *optional*) – File format to save the data. Defaults to 'json'.
- **n\_jobs** (*int*, *optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore\_exceptions** (*bool*, *optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use\_converted** (*bool*, *optional*) – Force to disable on-the-fly mode and use stored converted data

See also:

**muspy.FolderDataset** Class for datasets storing files in a folder.

**muspy.RemoteDataset** Base class for remote MusPy datasets.

**read** (*filename: str*) → `muspy.music.Music`  
Read a file into a Music object.

```
class muspy.datasets.RemoteMusicDataset (root: Union[str, pathlib.Path], download_and_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, kind: str = 'json', verbose: bool = True)
```

Base class for remote datasets of MusPy JSON/YAML files.

**root**  
Root directory of the dataset.

**Type** `str` or `Path`

**kind**  
File format of the data. Defaults to 'json'.

**Type** {'json', 'yaml'}, optional

#### Parameters

- **download\_and\_extract** (*bool*, *optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool*, *optional*) – Whether to remove the source archive(s). Defaults to False.

See also:

**muspy.MusicDataset** Class for datasets of MusPy JSON/YAML files.

**muspy.RemoteDataset** Base class for remote MusPy datasets.

```
class muspy.datasets.WikifoniaDataset (root: Union[str, pathlib.Path], download_and_extract: bool = False, overwrite: Optional[bool] = None, cleanup: Optional[bool] = None, convert: Optional[bool] = None, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: Optional[bool] = None, use_converted: Optional[bool] = None, verbose: bool = True)
```

Wikifonia dataset.

**read** (filename: Union[str, pathlib.Path]) → muspy.music.Music  
Read a file into a Music object.

**muspy.datasets.get\_dataset** (key: str) → Type[muspy.datasets.base.Dataset]  
Return a certain dataset class by key.

**Parameters** **key** (str) – Dataset key (case-insensitive).

**Returns**

**Return type** The corresponding dataset class.

**muspy.datasets.list\_datasets** ()  
Return all supported dataset classes as a list.

**Returns**

**Return type** A list of all supported dataset classes.

### 7.10.3 muspy.external

External dependencies.

This module provides functions for working with external dependencies.

#### Functions

- download\_bravura\_font
- download\_musescore\_soundfont
- get\_bravura\_font\_dir
- get\_bravura\_font\_path
- get\_musescore\_soundfont\_dir
- get\_musescore\_soundfont\_path

**muspy.external.download\_bravura\_font** ()  
Download the Bravura font.

**muspy.external.download\_musescore\_soundfont** ()  
Download the MuseScore General soundfont.

**muspy.external.get\_bravura\_font\_dir** () → pathlib.Path  
Return path to the directory of the Bravura font.

**muspy.external.get\_bravura\_font\_path** () → pathlib.Path  
Return path to the Bravura font.

**muspy.external.get\_musescore\_soundfont\_dir** () → pathlib.Path  
Return path to the MuseScore General soundfont directory.

---

`muspy.external.get_musescore_soundfont_path()` → `pathlib.Path`  
Return path to the MuseScore General soundfont.

## 7.10.4 muspy.inputs

Input interfaces.

This module provides input interfaces for common symbolic music formats, MusPy's native JSON and YAML formats, other symbolic music libraries and commonly-used representations in music generation.

### Functions

- `from_event_representation`
- `from_mido`
- `from_music21`
- `from_music21_opus`
- `from_note_representation`
- `from_object`
- `from_pianoroll_representation`
- `from_pitch_representation`
- `from_pretty_midi`
- `from_pypianoroll`
- `from_representation`
- `load`
- `load_json`
- `load_yaml`
- `read`
- `read_abc`
- `read_abc_string`
- `read_midi`
- `read_musicxml`

### Errors

- `MIDIError`
- `MusicXMLError`

**exception** `muspy.inputs.MIDIError`  
An error class for MIDI related exceptions.

**exception** `muspy.inputs.MusicXMLError`  
An error class for MusicXML related exceptions.

```
muspy.inputs.from_event_representation(array: numpy.ndarray, resolution: int = 24,  
                                       program: int = 0, is_drum: bool = False,  
                                       use_single_note_off_event: bool = False,  
                                       use_end_of_sequence_event: bool = False,  
                                       max_time_shift: int = 100, velocity_bins: int = 32,  
                                       default_velocity: int = 64, duplicate_note_mode: str  
                                       = 'fifo') → muspy.music.Music
```

Decode event-based representation into a Music object.

#### Parameters

- **array** (*ndarray*) – Array in event-based representation to decode. Cast to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.
- **program** (*int*, *optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool*, *optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_single\_note\_off\_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, a note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use\_end\_of\_sequence\_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **max\_time\_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max\_time\_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity\_bins** (*int*) – Number of velocity bins to use. Defaults to 32.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.
- **duplicate\_note\_mode** (*{'fifo', 'lifo', 'close\_all'}*) – Policy for dealing with duplicate notes. When a note off event is presetned while there are multiple corresponding note on events that have not yet been closed, we need a policy to decide which note on messages to close. This is only effective when *use\_single\_note\_off\_event* is False. Defaults to 'fifo'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out): close the latest note on
  - 'close\_all': close all note on messages

**Returns** Decoded Music object.

**Return type** *muspy.Music*

#### References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

```
muspy.inputs.from_mido(midi: mido.midifiles.midifiles.MidiFile, duplicate_note_mode: str = 'fifo')  
→ muspy.music.Music
```

Return a mido MidiFile object as a Music object.



**Parameters**

- **midi** (`mido.MidiFile`) – Mido MidiFile object to convert.
- **duplicate\_note\_mode** (`{'fifo', 'lifo', 'close_all'}`) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspodng note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out): close the latest note on
  - 'close\_all': close all note on messages

**Returns** Converted Music object.

**Return type** `muspy.Music`

```
muspy.inputs.from_music21 (stream: music21.stream.Stream, resolution=24) →
                             Union[muspy.music.Music, List[muspy.music.Music],
                             muspy.classes.Track, List[muspy.classes.Track]]
```

Return a music21 Stream object as Music or Track object(s).

**Parameters**

- **stream** (`music21.stream.Stream`) – Stream object to convert.
- **resolution** (`int, optional`) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Converted Music or Track object(s).

**Return type** `muspy.Music` or `muspy.Track`

```
muspy.inputs.from_music21_opus (opus: music21.stream.Opus, resolution=24) →
                                List[muspy.music.Music]
```

Return a music21 Opus object as a list of Music objects.

**Parameters**

- **opus** (`music21.stream.Opus`) – Opus object to convert.
- **resolution** (`int, optional`) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Converted Music object.

**Return type** `muspy.Music`

```
muspy.inputs.from_music21_part (part: music21.stream.Part, resolution=24) →
                                Union[muspy.classes.Track, List[muspy.classes.Track]]
```

Return a music21 Part object as Track object(s).

**Parameters**

- **part** (`music21.stream.Part`) – Part object to parse.
- **resolution** (`int, optional`) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Parsed track(s).

**Return type** `muspy.Track` or list of `muspy.Track`

```
muspy.inputs.from_music21_score (score: music21.stream.Score, resolution=24) →
                                muspy.music.Music
```

Return a music21 Stream object as a Music object.

**Parameters**

- **score** (*music21.stream.Score*) – Score object to convert.
- **resolution** (*int*, *optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.

**Returns** Converted Music object.

**Return type** *muspy.Music*

`muspy.inputs.from_note_representation` (*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is\_drum: bool = False*, *use\_start\_end: bool = False*, *encode\_velocity: bool = True*, *default\_velocity: int = 64*) → *muspy.music.Music*

Decode note-based representation into a Music object.

**Parameters**

- **array** (*ndarray*) – Array in note-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.
- **program** (*int*, *optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool*, *optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_start\_end** (*bool*) – Whether to use ‘start’ and ‘end’ to encode the timing rather than ‘time’ and ‘duration’. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **default\_velocity** (*int*) – Default velocity value to use when decoding if *encode\_velocity* is False. Defaults to 64.

**Returns** Decoded Music object.

**Return type** *muspy.Music*

**References**

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

`muspy.inputs.from_object` (*obj: Union[music21.stream.Stream, mido.midifiles.midifiles.MidiFile, pretty\_midi.pretty\_midi.PrettyMIDI, pypianoroll.multitrack.Multitrack]*, *\*\*kwargs*) → *Union[muspy.music.Music, List[muspy.music.Music], muspy.classes.Track, List[muspy.classes.Track]]*

Return an outside object as a Music object.

**Parameters**

- **obj** – Object to convert. Supported objects are *music21.Stream*, *mido.MidiTrack*, *pretty\_midi.PrettyMIDI*, and *pypianoroll.Multitrack* objects.
- **\*\*kwargs** – Keyword arguments to pass to *muspy.from\_music21()*, *muspy.from\_mido()*, *from\_pretty\_midi()* or *from\_pypianoroll()*.

**Returns** Converted Music object.

**Return type** *muspy.Music*

`muspy.inputs.from_pianoroll_representation` (*array*: *numpy.ndarray*, *resolution*: *int* = 24, *program*: *int* = 0, *is\_drum*: *bool* = False, *encode\_velocity*: *bool* = True, *default\_velocity*: *int* = 64) → `muspy.music.Music`

Decode pitch-based representation into a Music object.

#### Parameters

- **array** (*ndarray*) – Array in piano-roll representation to decode. Will be casted to integer if not of integer type. If *encode\_velocity* is True, will be casted to boolean if not of boolean type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.
- **program** (*int*, *optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool*, *optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode velocities. Defaults to True.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** Decoded Music object.

**Return type** `muspy.Music`

#### References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

`muspy.inputs.from_pitch_representation` (*array*: *numpy.ndarray*, *resolution*: *int* = 24, *program*: *int* = 0, *is\_drum*: *bool* = False, *use\_hold\_state*: *bool* = False, *default\_velocity*: *int* = 64) → `muspy.music.Music`

Decode pitch-based representation into a Music object.

#### Parameters

- **array** (*ndarray*) – Array in pitch-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.
- **program** (*int*, *optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is\_drum** (*bool*, *optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use\_hold\_state** (*bool*) – Whether to use a special state for holds. Defaults to False.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** Decoded Music object.

**Return type** `muspy.Music`

## References

[1] <https://www.midi.org/specifications/item/gm-level-1-sound-set>

`muspy.inputs.from_pretty_midi` (*midi*: `pretty_midi.pretty_midi.PrettyMIDI`, *resolution*: *Optional[int] = None*) → `muspy.music.Music`

Return a `pretty_midi` `PrettyMIDI` object as a `Music` object.

### Parameters

- **midi** (`pretty_midi.PrettyMIDI`) – `PrettyMIDI` object to convert.
- **resolution** (*int*, *optional*) – Time steps per quarter note. Defaults to `muspy.DEFAULT_RESOLUTION`.

**Returns** Converted `Music` object.

**Return type** `muspy.Music`

`muspy.inputs.from_pypianoroll` (*multitrack*: `pypianoroll.multitrack.Multitrack`, *default\_velocity*: *int = 64*) → `muspy.music.Music`

Return a `Pypianoroll` `Multitrack` object as a `Music` object.

### Parameters

- **multitrack** (`pypianoroll.Multitrack`) – `Pypianoroll` `Multitrack` object to convert.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** `music` – Converted `MusPy` `Music` object.

**Return type** `muspy.Music`

`muspy.inputs.from_pypianoroll_track` (*track*: `pypianoroll.track.Track`, *default\_velocity*: *int = 64*) → `muspy.classes.Track`

Return a `Pypianoroll` `Track` object as a `Track` object.

### Parameters

- **track** (`pypianoroll.Track`) – `Pypianoroll` `Track` object to convert.
- **default\_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

**Returns** Converted track.

**Return type** `muspy.Track`

`muspy.inputs.from_representation` (*array*: `numpy.ndarray`, *kind*: *str*, *\*\*kwargs*) → `muspy.music.Music`

Update with the given representation.

### Parameters

- **array** (`numpy.ndarray`) – Array in a supported representation.
- **kind** (*str*, {'pitch', 'pianoroll', 'event', 'note'}) – Data representation.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.from_pitch_representation()`, `muspy.from_pianoroll_representation()`, `from_event_representation()` or `from_note_representation()`.

**Returns** Converted `Music` object.

**Return type** `muspy.Music`

`muspy.inputs.load(path: Union[str, pathlib.Path, TextIO], kind: Optional[str] = None, **kwargs) → muspy.music.Music`

Load a JSON or a YAML file into a Music object.

This is a wrapper function for `muspy.load_json()` and `muspy.load_yaml()`.

#### Parameters

- **path** (`str`, `Path` or `TextIO`) – Path to the file or the file to load.
- **kind** (`{'json', 'yaml'}`, `optional`) – Format to save. Defaults to infer from the extension.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.load_json()` or `muspy.load_yaml()`.

**Returns** Loaded Music object.

**Return type** `muspy.Music`

See also:

`muspy.load_json()` Load a JSON file into a Music object.

`muspy.load_yaml()` Load a YAML file into a Music object.

`muspy.read()` Read a MIDI/MusicXML/ABC file into a Music object.

`muspy.inputs.load_json(path: Union[str, pathlib.Path, TextIO], compressed: Optional[bool] = None) → muspy.music.Music`

Load a JSON file into a Music object.

#### Parameters

- **path** (`str`, `Path` or `TextIO`) – Path to the file or the file to load.
- **compressed** (`bool`, `optional`) – Whether the file is a compressed JSON file (`.json.gz`). Has no effect when `path` is a file object. Defaults to infer from the extension (`.gz`).

**Returns** Loaded Music object.

**Return type** `muspy.Music`

#### Notes

When a path is given, assume UTF-8 encoding and gzip compression if `compressed=True`.

`muspy.inputs.load_yaml(path: Union[str, pathlib.Path, TextIO], compressed: Optional[bool] = None) → muspy.music.Music`

Load a YAML file into a Music object.

#### Parameters

- **path** (`str`, `Path` or `TextIO`) – Path to the file or the file to load.
- **compressed** (`bool`, `optional`) – Whether the file is a compressed YAML file (`.yaml.gz`). Has no effect when `path` is a file object. Defaults to infer from the extension (`.gz`).

**Returns** Loaded Music object.

**Return type** `muspy.Music`

## Notes

When a path is given, assume UTF-8 encoding and gzip compression if *compressed=True*.

```
muspy.inputs.read(path: Union[str, pathlib.Path], kind: Optional[str] = None, **kwargs) →  
Union[muspy.music.Music, List[muspy.music.Music]]
```

Read a MIDI/MusicXML/ABC file into a Music object.

### Parameters

- **path** (*str* or *Path*) – Path to the file to read.
- **kind** (*{'midi', 'musicxml', 'abc'}*, *optional*) – Format to save. Defaults to infer from the extension.
- **\*\*kwargs** – Keyword arguments to pass to *muspy.read\_midi()*, *muspy.read\_musicxml()* or *read\_abc()*.

**Returns** Converted Music object(s).

**Return type** *muspy.Music* or list of *muspy.Music*

See also:

*muspy.load()* Load a JSON or a YAML file into a Music object.

```
muspy.inputs.read_abc(path: Union[str, pathlib.Path], number: Optional[int] = None, resolution=24) → Union[muspy.music.Music, List[muspy.music.Music]]
```

Return an ABC file into Music object(s) using music21 backend.

### Parameters

- **path** (*str* or *Path*) – Path to the ABC file to read.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the ‘X:’ field).
- **resolution** (*int*, *optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.

**Returns** Converted Music object(s).

**Return type** list of *muspy.Music*

```
muspy.inputs.read_abc_string(data_str: str, number: Optional[int] = None, resolution=24) →  
Union[muspy.music.Music, List[muspy.music.Music]]
```

Read ABC data into Music object(s) using music21 backend.

### Parameters

- **data\_str** (*str*) – ABC data to parse.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the ‘X:’ field).
- **resolution** (*int*, *optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT\_RESOLUTION*.

**Returns** Converted Music object(s).

**Return type** *muspy.Music*

```
muspy.inputs.read_midi(path: Union[str, pathlib.Path], backend: str = 'mido', duplicate_note_mode:  
str = 'fifo') → muspy.music.Music
```

Read a MIDI file into a Music object.

### Parameters

- **path** (*str* or *Path*) – Path to the MIDI file to read.

- **backend** ({'mido', 'pretty\_midi'}) – Backend to use.
- **duplicate\_note\_mode** ({'fifo', 'lifo', 'close\_all'}) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspodng note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'. Only used when *backend*='mido'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out):close the latest note on
  - 'close\_all': close all note on messages

**Returns** Converted Music object.

**Return type** *muspy.Music*

`muspy.inputs.read_musicxml (path: Union[str, pathlib.Path], compressed: Optional[bool] = None)`  
     → `muspy.music.Music`

Read a MusicXML file into a Music object.

**Parameters** *path* (*str* or *Path*) – Path to the MusicXML file to read.

**Returns** Converted Music object.

**Return type** *muspy.Music*

## Notes

Grace notes and unpitched notes are not supported.

## 7.10.5 muspy.metrics

Objective metrics.

This module provides common objective metrics in music generation. These objective metrics could be used to evaluate a music generation system by comparing the statistical difference between the training data and the generated samples.

## Functions

- `drum_in_pattern_rate`
- `drum_pattern_consistency`
- `empty_beat_rate`
- `empty_measure_rate`
- `groove_consistency`
- `n_pitch_classes_used`
- `n_pitches_used`
- `pitch_class_entropy`
- `pitch_entropy`
- `pitch_in_scale_rate`
- `pitch_range`

- polyphony
- polyphony\_rate
- scale\_consistency

`muspy.metrics.drum_in_pattern_rate` (*music*: *muspy.music.Music*, *meter*: *str*) → float

Return the ratio of drum notes in a certain drum pattern.

The drum-in-pattern rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Only drum tracks are considered. Return NaN if no drum note is found. This metric is used in [1].

$$\text{drum\_in\_pattern\_rate} = \frac{\#(\text{drum\_notes\_in\_pattern})}{\#(\text{drum\_notes})}$$

#### Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **meter** (*str*, {'duple', 'triple'}) – Meter of the drum pattern.

**Returns** Drum-in-pattern rate.

**Return type** float

See also:

[`muspy.drum\_pattern\_consistency\(\)`](#) Compute the largest drum-in-pattern rate.

#### References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.metrics.drum_pattern_consistency` (*music*: *muspy.music.Music*) → float

Return the largest drum-in-pattern rate.

The drum pattern consistency is defined as the largest drum-in-pattern rate over duple and triple meters. Only drum tracks are considered. Return NaN if no drum note is found.

$$\text{drum\_pattern\_consistency} = \max_{\text{meter}} \text{drum\_in\_pattern\_rate}(\text{meter})$$

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Drum pattern consistency.

**Return type** float

See also:

[`muspy.drum\_in\_pattern\_rate\(\)`](#) Compute the ratio of drum notes in a certain drum pattern.

`muspy.metrics.empty_beat_rate` (*music*: *muspy.music.Music*) → float

Return the ratio of empty beats.

The empty-beat rate is defined as the ratio of the number of empty beats (where no note is played) to the total number of beats. Return NaN if song length is zero. This metric is also implemented in Pypianoroll [1].

$$\text{empty\_beat\_rate} = \frac{\#(\text{empty\_beats})}{\#(\text{beats})}$$



**Parameters** `music` (*muspy.Music*) – Music object to evaluate.

**Returns** Empty-beat rate.

**Return type** `float`

See also:

`muspy.empty_measure_rate()` Compute the ratio of empty measures.

## References

1. Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang, “Pypianoroll: Open Source Python Package for Handling Multitrack Pianorolls,” in Late-Breaking Demos of the 18th International Society for Music Information Retrieval Conference (ISMIR), 2018.

`muspy.metrics.empty_measure_rate` (*music: muspy.music.Music, measure\_resolution: int*) → `float`

Return the ratio of empty measures.

The empty-measure rate is defined as the ratio of the number of empty measures (where no note is played) to the total number of measures. Note that this metric only works for songs with a constant time signature. Return NaN if song length is zero. This metric is used in [1].

$$\text{empty\_measure\_rate} = \frac{\#(\text{empty\_measures})}{\#(\text{measures})}$$

### Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure\_resolution** (*int*) – Time steps per measure.

**Returns** Empty-measure rate.

**Return type** `float`

See also:

`muspy.empty_beat_rate()` Compute the ratio of empty beats.

## References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.metrics.groove_consistency` (*music: muspy.music.Music, measure\_resolution: int*) → `float`

Return the groove consistency.

The groove consistency is defined as the mean hamming distance of the neighboring measures.

$$\text{groove\_consistency} = 1 - \frac{1}{T-1} \sum_{i=1}^{T-1} d(G_i, G_{i+1})$$

Here,  $T$  is the number of measures,  $G_i$  is the binary onset vector of the  $i$ -th measure (a one at position that has an onset, otherwise a zero), and  $d(G, G')$  is the hamming distance between two vectors  $G$  and  $G'$ . Note that this metric only works for songs with a constant time signature. Return NaN if the number of measures is less than two. This metric is used in [1].

**Parameters**

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure\_resolution** (*int*) – Time steps per measure.

**Returns** Groove consistency.

**Return type** *float*

**References**

1. Shih-Lun Wu and Yi-Hsuan Yang, “The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures”, in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

`muspy.metrics.n_pitch_classes_used(music: muspy.music.Music) → int`

Return the number of unique pitch classes used.

Drum tracks are ignored.

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Number of unique pitch classes used.

**Return type** *int*

**See also:**

`muspy.n_pitches_used()` Compute the number of unique pitches used.

`muspy.metrics.n_pitches_used(music: muspy.music.Music) → int`

Return the number of unique pitches used.

Drum tracks are ignored.

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Number of unique pitch used.

**Return type** *int*

**See also:**

`muspy.n_pitch_class_used()` Compute the number of unique pitch classes used.

`muspy.metrics.pitch_class_entropy(music: muspy.music.Music) → float`

Return the entropy of the normalized note pitch class histogram.

The pitch class entropy is defined as the Shannon entropy of the normalized note pitch class histogram. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$\text{pitch\_class\_entropy} = - \sum_{i=0}^{11} P(\text{pitch\_class} = i) \times \log_2 P(\text{pitch\_class} = i)$$

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Pitch class entropy.

**Return type** *float*

**See also:**

**`muspy.pitch_entropy()`** Compute the entropy of the normalized pitch histogram.

## References

1. Shih-Lun Wu and Yi-Hsuan Yang, “The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures”, in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

`muspy.metrics.pitch_entropy(music: muspy.music.Music) → float`

Return the entropy of the normalized note pitch histogram.

The pitch entropy is defined as the Shannon entropy of the normalized note pitch histogram. Drum tracks are ignored. Return NaN if no note is found.

$$pitch\_entropy = - \sum_{i=0}^{127} P(pitch = i) \log_2 P(pitch = i)$$

**Parameters** **music** (`muspy.Music`) – Music object to evaluate.

**Returns** Pitch entropy.

**Return type** `float`

See also:

**`muspy.pitch_class_entropy()`** Compute the entropy of the normalized pitch class histogram.

`muspy.metrics.pitch_in_scale_rate(music: muspy.music.Music, root: int, mode: str) → float`

Return the ratio of pitches in a certain musical scale.

The pitch-in-scale rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$pitch\_in\_scale\_rate = \frac{\#(notes\_in\_scale)}{\#(notes)}$$

**Parameters**

- **music** (`muspy.Music`) – Music object to evaluate.
- **root** (`int`) – Root of the scale.
- **mode** (`str`, `{'major', 'minor'}`) – Mode of the scale.

**Returns** Pitch-in-scale rate.

**Return type** `float`

See also:

**`muspy.scale_consistency()`** Compute the largest pitch-in-class rate.

## References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.metrics.pitch_range(music: muspy.music.Music) → int`

Return the pitch range.

Drum tracks are ignored. Return zero if no note is found.

**Parameters** `music` (*`muspy.Music`*) – Music object to evaluate.

**Returns** Pitch range.

**Return type** `int`

`muspy.metrics.polyphony(music: muspy.music.Music) → float`

Return the average number of pitches being played concurrently.

The polyphony is defined as the average number of pitches being played at the same time, evaluated only at time steps where at least one pitch is on. Drum tracks are ignored. Return NaN if no note is found.

$$\text{polyphony} = \frac{\#(\text{pitches\_when\_at\_least\_one\_pitch\_is\_on})}{\#(\text{time\_steps\_where\_at\_least\_one\_pitch\_is\_on})}$$

**Parameters** `music` (*`muspy.Music`*) – Music object to evaluate.

**Returns** Polyphony.

**Return type** `float`

See also:

*`muspy.polyphony_rate()`* Compute the ratio of time steps where multiple pitches are on.

`muspy.metrics.polyphony_rate(music: muspy.music.Music, threshold: int = 2) → float`

Return the ratio of time steps where multiple pitches are on.

The polyphony rate is defined as the ratio of the number of time steps where multiple pitches are on to the total number of time steps. Drum tracks are ignored. Return NaN if song length is zero. This metric is used in [1], where it is called *polyphonicity*.

$$\text{polyphony\_rate} = \frac{\#(\text{time\_steps\_where\_multiple\_pitches\_are\_on})}{\#(\text{time\_steps})}$$

**Parameters**

- **music** (*`muspy.Music`*) – Music object to evaluate.
- **threshold** (*`int`*) – Threshold of number of pitches to count into the numerator.

**Returns** Polyphony rate.

**Return type** `float`

See also:

*`muspy.polyphony()`* Compute the average number of pitches being played at the same time.

## References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`muspy.metrics.scale_consistency` (*music*: *muspy.music.Music*) → float

Return the largest pitch-in-scale rate.

The scale consistency is defined as the largest pitch-in-scale rate over all major and minor scales. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$scale\_consistency = \max_{root, mode} pitch\_in\_scale\_rate(root, mode)$$

**Parameters** **music** (*muspy.Music*) – Music object to evaluate.

**Returns** Scale consistency.

**Return type** float

**See also:**

`muspy.pitch_in_scale_rate()` Compute the ratio of pitches in a certain musical scale.

## References

1. Olof Mogren, “C-RNN-GAN: Continuous recurrent neural networks with adversarial training,” in NeuIPS Workshop on Constructive Machine Learning, 2016.

## 7.10.6 muspy.outputs

Output interfaces.

This module provides output interfaces for common symbolic music formats, MusPy’s native JSON and YAML formats, other symbolic music libraries and commonly-used representations in music generation.

### Functions

- `save`
- `save_json`
- `save_yaml`
- `to_event_representation`
- `to_mido`
- `to_music21`
- `to_note_representation`
- `to_object`
- `to_pianoroll_representation`
- `to_pitch_representation`
- `to_pretty_midi`
- `to_pypianoroll`
- `to_representation`
- `write`
- `write_abc`

- `write_audio`
- `write_midi`
- `write_musicxml`

`muspy.outputs.save` (*path*: Union[str, pathlib.Path, TextIO], *music*: Music, *kind*: Optional[str] = None, *\*\*kwargs*)

Save a Music object loselessly to a JSON or a YAML file.

This is a wrapper function for `muspy.save_json()` and `muspy.save_yaml()`.

#### Parameters

- **path** (*str*, Path or TextIO) – Path or file to save the data.
- **music** (*muspy.Music*) – Music object to save.
- **kind** ({'json', 'yaml'}, optional) – Format to save. Defaults to infer from the extension.
- **\*\*kwargs** – Keyword arguments to pass to `muspy.save_json()` or `muspy.save_yaml()`.

See also:

`muspy.save_json()` Save a Music object to a JSON file.

`muspy.save_yaml()` Save a Music object to a YAML file.

`muspy.write()` Write a Music object to a MIDI/MusicXML/ABC/audio file.

#### Notes

The conversion can be lossy if any nonserializable object is used (for example, an Annotation object, which can store data of any type).

`muspy.outputs.save_json` (*path*: Union[str, pathlib.Path, TextIO], *music*: Music, *skip\_missing*: bool = True, *ensure\_ascii*: bool = False, *compressed*: Optional[bool] = None, *\*\*kwargs*)

Save a Music object to a JSON file.

#### Parameters

- **path** (*str*, Path or TextIO) – Path or file to save the JSON data.
- **music** (*muspy.Music*) – Music object to save.
- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **ensure\_ascii** (*bool*) – Whether to escape non-ASCII characters. Will be passed to PyYAML's `yaml.dump`. Defaults to False.
- **compressed** (*bool*, optional) – Whether to save as a compressed JSON file (`.json.gz`). Has no effect when *path* is a file object. Defaults to infer from the extension (`.gz`).
- **\*\*kwargs** – Keyword arguments to pass to `json.dumps()`.

## Notes

When a path is given, use UTF-8 encoding and gzip compression if *compressed=True*.

```
muspy.outputs.save_yaml(path: Union[str, pathlib.Path, TextIO], music: Music, skip_missing: bool = True, allow_unicode: bool = True, compressed: Optional[bool] = None, **kwargs)
```

Save a Music object to a YAML file.

### Parameters

- **path** (*str*, *Path* or *TextIO*) – Path or file to save the YAML data.
- **music** (*muspy.Music*) – Music object to save.
- **skip\_missing** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.
- **allow\_unicode** (*bool*) – Whether to escape non-ASCII characters. Will be passed to `json.dumps()`. Defaults to False.
- **compressed** (*bool*, *optional*) – Whether to save as a compressed YAML file (*.yaml.gz*). Has no effect when *path* is a file object. Defaults to infer from the extension (*.gz*).
- **\*\*kwargs** – Keyword arguments to pass to *yaml.dump*.

## Notes

When a path is given, use UTF-8 encoding and gzip compression if *compressed=True*.

```
muspy.outputs.to_event_representation(music: Music, use_single_note_off_event: bool = False, use_end_of_sequence_event: bool = False, encode_velocity: bool = False, force_velocity_event: bool = True, max_time_shift: int = 100, velocity_bins: int = 32) → numpy.ndarray
```

Encode a Music object into event-based representation.

The event-based representation represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is *M* x 1, where *M* is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-on events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

### Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use\_single\_note\_off\_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use\_end\_of\_sequence\_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode velocities.
- **force\_velocity\_event** (*bool*) – Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.

- **max\_time\_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max\_time\_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity\_bins** (*int*) – Number of velocity bins to use. Defaults to 32.

**Returns** Encoded array in event-based representation.

**Return type** ndarray, dtype=uint16, shape=(?, 1)

`muspy.outputs.to_mido` (*music: Music, use\_note\_off\_message: bool = False*)  
Return a Music object as a MidiFile object.

**Parameters**

- **music** (*muspy.Music* object) – Music object to convert.
- **use\_note\_off\_message** (*bool, optional*) – Whether to use note-off messages. If False, note-on messages with zero velocity are used instead. The advantage to using note-on messages at zero velocity is that it can avoid sending additional status bytes when Running Status is employed. Defaults to False.

**Returns** Converted MidiFile object.

**Return type** `mido.MidiFile`

`muspy.outputs.to_music21` (*music: Music*) → `music21.stream.Score`  
Return a Music object as a music21 Score object.

**Parameters** **music** (*muspy.Music*) – Music object to convert.

**Returns** Converted music21 Score object.

**Return type** `music21.stream.Score`

`muspy.outputs.to_note_representation` (*music: Music, use\_start\_end: bool = False, encode\_velocity: bool = True, dtype: Union[numpy.dtype, type, str] = <class 'int'>*) → `numpy.ndarray`

Encode a Music object into note-based representation.

The note-based representation represents music as a sequence of (time, pitch, duration, velocity) tuples. For example, a note `Note(time=0, duration=4, pitch=60, velocity=64)` will be encoded as a tuple `(0, 60, 4, 64)`. The output shape is `N * D`, where `N` is the number of notes and `D` is 4 when *encode\_velocity* is True, otherwise `D` is 3. The values of the second dimension represent time, pitch, duration and velocity (discarded when *encode\_velocity* is False).

**Parameters**

- **music** (*muspy.Music*) – Music object to encode.
- **use\_start\_end** (*bool*) – Whether to use ‘start’ and ‘end’ to encode the timing rather than ‘time’ and ‘duration’. Defaults to False.
- **encode\_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **dtype** (*dtype, type or str*) – Data type of the return array. Defaults to int.

**Returns** Encoded array in note-based representation.

**Return type** ndarray, shape=(?, 3 or 4)

`muspy.outputs.to_object` (*music: Music, kind: str, \*\*kwargs*) → `Union[music21.stream.Stream, mido.midifiles.midifiles.MidiFile, pretty_midi.pretty_midi.PrettyMIDI, pypianoroll.multitrack.Multitrack]`  
Return a Music object as an object in other libraries.



Supported classes are `music21.Stream`, `mido.MidiTrack`, `pretty_midi.PrettyMIDI` and `pypianoroll.Multitrack`.

#### Parameters

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str*, {'music21', 'mido', 'pretty\_midi', 'pypianoroll'}) – Target class.

**Returns** Converted object.

**Return type** `music21.Stream`, `mido.MidiTrack`, `pretty_midi.PrettyMIDI` or `pypianoroll.Multitrack`

`muspy.outputs.to_pianoroll_representation` (*music: Music, encode\_velocity: bool = True*)  
→ `numpy.ndarray`

Encode notes into piano-roll representation.

#### Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **encode\_velocity** (*bool*) – Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

**Returns** Encoded array in piano-roll representation.

**Return type** `ndarray`, `dtype=uint8` or `bool`, `shape=(?, 128)`

`muspy.outputs.to_pitch_representation` (*music: Music, use\_hold\_state: bool = False*) →  
`numpy.ndarray`

Encode a Music object into pitch-based representation.

The pitch-based representation represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is T x 1, where T is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

#### Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use\_hold\_state** (*bool*) – Whether to use a special state for holds. Defaults to False.

**Returns** Encoded array in pitch-based representation.

**Return type** `ndarray`, `dtype=uint8`, `shape=(?, 1)`

`muspy.outputs.to_pretty_midi` (*music: Music*) → `pretty_midi.pretty_midi.PrettyMIDI`

Return a Music object as a PrettyMIDI object.

Tempo changes are not supported yet.

**Parameters** **music** (*muspy.Music* object) – Music object to convert.

**Returns** Converted PrettyMIDI object.

**Return type** `pretty_midi.PrettyMIDI`

#### Notes

Tempo information will not be included in the output.

`muspy.outputs.to_pypianoroll` (*music: Music*) → `pypianoroll.multitrack.Multitrack`

Return a Music object as a Multitrack object.

**Parameters** `music` (*muspy.Music*) – Music object to convert.

**Returns** `multitrack` – Converted Multitrack object.

**Return type** `pypianoroll.Multitrack`

`muspy.outputs.to_representation` (*music: Music, kind: str, \*\*kwargs*) → `numpy.ndarray`

Return a Music object in a specific representation.

**Parameters**

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str*, {'pitch', 'piano-roll', 'event', 'note'}) – Target representation.

**Returns** `array` – Converted representation.

**Return type** `ndarray`

`muspy.outputs.synthesize` (*music: Music, soundfont\_path: Union[str, pathlib.Path, None] = None, rate: int = 44100*) → `numpy.ndarray`

Synthesize a Music object to raw audio.

**Parameters**

- **music** (*muspy.Music*) – Music object to write.
- **soundfont\_path** (*str or Path, optional*) – Path to the soundfont file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.

**Returns** Synthesized waveform.

**Return type** `ndarray, dtype=int16, shape=(?, 2)`

`muspy.outputs.write` (*path: Union[str, pathlib.Path], music: Music, kind: Optional[str] = None, \*\*kwargs*)

Write a Music object to a MIDI/MusicXML/ABC/audio file.

**Parameters**

- **path** (*str or Path*) – Path to write the file.
- **music** (*muspy.Music*) – Music object to convert.
- **kind** ({'midi', 'musicxml', 'abc', 'audio'}, *optional*) – Format to save. Defaults to infer from the extension.

**See also:**

`muspy.save()` Save a Music object loselessly to a JSON or a YAML file.

`muspy.outputs.write_abc` (*path: Union[str, pathlib.Path], music: Music*)

Write a Music object to a ABC file.

**Parameters**

- **path** (*str or Path*) – Path to write the ABC file.
- **music** (*muspy.Music*) – Music object to write.

`muspy.outputs.write_audio` (*path: Union[str, pathlib.Path], music: Music, soundfont\_path: Union[str, pathlib.Path, None] = None, rate: int = 44100, audio\_format: Optional[str] = None*)

Write a Music object to an audio file.

Supported formats include WAV, AIFF, FLAC and OGA.

#### Parameters

- **path** (*str* or *Path*) – Path to write the audio file.
- **music** (*muspy.Music*) – Music object to write.
- **soundfont\_path** (*str* or *Path*, *optional*) – Path to the soundfont file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.
- **audio\_format** (*str*, {'wav', 'aiff', 'flac', 'oga'}, *optional*) – File format to write. If None, infer it from the extension.

`muspy.outputs.write_midi(path: Union[str, pathlib.Path], music: Music, backend: str = 'mido', **kwargs)`

Write a Music object to a MIDI file.

#### Parameters

- **path** (*str* or *Path*) – Path to write the MIDI file.
- **music** (*muspy.Music*) – Music object to write.
- **backend** ({'mido', 'pretty\_midi'}, *optional*) – Backend to use. Defaults to 'mido'.

See also:

**write\_midi\_mido()** Write a Music object to a MIDI file using mido as backend.

**write\_midi\_pretty\_midi()** Write a Music object to a MIDI file using pretty\_midi as backend.

`muspy.outputs.write_musicxml(path: Union[str, pathlib.Path], music: Music, compressed: Optional[bool] = None)`

Write a Music object to a MusicXML file.

#### Parameters

- **path** (*str* or *Path*) – Path to write the MusicXML file.
- **music** (*muspy.Music*) – Music object to write.
- **compressed** (*bool*, *optional*) – Whether to write to a compressed MusicXML file. If None, infer from the extension of the filename ('.xml' and '.musicxml' for an uncompressed file, '.mxl' for a compressed file).

## 7.10.7 muspy.processors

Representation processors.

This module defines the processors for commonly used representations.

### Classes

- NoteRepresentationProcessor
- EventRepresentationProcessor
- PianoRollRepresentationProcessor

- PitchRepresentationProcessor

```
class muspy.processors.NoteRepresentationProcessor(use_start_end: bool = False, en-  
code_velocity: bool = True, dtype:  
Union[numpy.dtype, type, str] =  
<class 'int'>, default_velocity: int  
= 64)
```

Note-based representation processor.

The note-based representation represents music as a sequence of (pitch, time, duration, velocity) tuples. For example, a note `Note(time=0, duration=4, pitch=60, velocity=64)` will be encoded as a tuple `(0, 4, 60, 64)`. The output shape is `L * D`, where `L` is the number of notes and `D` is 4 when `encode_velocity` is `True`, otherwise `D` is 3. The values of the second dimension represent pitch, time, duration and velocity (discarded when `encode_velocity` is `False`).

**use\_start\_end**

Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to `False`.

**Type** `bool`

**encode\_velocity**

Whether to encode note velocities. Defaults to `True`.

**Type** `bool`

**dtype**

Data type of the return array. Defaults to `int`.

**Type** `dtype, type` or `str`

**default\_velocity**

Default velocity value to use when decoding if `encode_velocity` is `False`. Defaults to 64.

**Type** `int`

**decode** (*array: numpy.ndarray*) → `muspy.music.Music`

Decode note-based representation into a Music object.

**Parameters** **array** (*ndarray*) – Array in note-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** `muspy.Music` object

**See also:**

`muspy.from_note_representation()` Return a Music object converted from note-based representation.

**encode** (*music: muspy.music.Music*) → `numpy.ndarray`

Encode a Music object into note-based representation.

**Parameters** **music** (`muspy.Music` object) – Music object to encode.

**Returns** Encoded array in note-based representation.

**Return type** `ndarray (np.uint8)`

**See also:**

`muspy.to_note_representation()` Convert a Music object into note-based representation.

```

class muspy.processors.EventRepresentationProcessor (use_single_note_off_event:
                                                    bool = False,
                                                    use_end_of_sequence_event:
                                                    bool = False,
                                                    encode_velocity: bool = False,
                                                    force_velocity_event: bool =
                                                    True,
                                                    max_time_shift: int =
                                                    100,
                                                    velocity_bins: int = 32,
                                                    default_velocity: int = 64)

```

Event-based representation processor.

The event-based representation represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is  $M \times 1$ , where  $M$  is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-on events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

#### **use\_single\_note\_off\_event**

Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.

**Type** `bool`

#### **use\_end\_of\_sequence\_event**

Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.

**Type** `bool`

#### **encode\_velocity**

Whether to encode velocities.

**Type** `bool`

#### **force\_velocity\_event**

Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.

**Type** `bool`

#### **max\_time\_shift**

Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max\_time\_shift* will be decomposed into two or more time-shift events. Defaults to 100.

**Type** `int`

#### **velocity\_bins**

Number of velocity bins to use. Defaults to 32.

**Type** `int`

#### **default\_velocity**

Default velocity value to use when decoding. Defaults to 64.

**Type** `int`

#### **decode** (*array: numpy.ndarray*) → *muspy.music.Music*

Decode event-based representation into a Music object.

**Parameters** *array* (*ndarray*) – Array in event-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** *muspy.Music* object

See also:

**`muspy.from_event_representation()`** Return a Music object converted from event-based representation.

**encode** (*music*: *muspy.music.Music*) → *numpy.ndarray*  
Encode a Music object into event-based representation.

**Parameters** **music** (*muspy.Music* object) – Music object to encode.

**Returns** Encoded array in event-based representation.

**Return type** *ndarray* (*np.uint16*)

See also:

**`muspy.to_event_representation()`** Convert a Music object into event-based representation.

**class** *muspy.processors.PianoRollRepresentationProcessor* (*encode\_velocity*: *bool* = *True*, *default\_velocity*: *int* = *64*)

Piano-roll representation processor.

The piano-roll representation represents music as a time-pitch matrix, where the columns are the time steps and the rows are the pitches. The values indicate the presence of pitches at different time steps. The output shape is *T* x 128, where *T* is the number of time steps.

**encode\_velocity**

Whether to encode velocities. If *True*, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to *True*.

**Type** *bool*

**default\_velocity**

Default velocity value to use when decoding if *encode\_velocity* is *False*. Defaults to 64.

**Type** *int*

**decode** (*array*: *numpy.ndarray*) → *muspy.music.Music*  
Decode piano-roll representation into a Music object.

**Parameters** **array** (*ndarray*) – Array in piano-roll representation to decode. Cast to integer if not of integer type. If *encode\_velocity* is *True*, casted to boolean if not of boolean type.

**Returns** Decoded Music object.

**Return type** *muspy.Music* object

See also:

**`muspy.from_pianoroll_representation()`** Return a Music object converted from piano-roll representation.

**encode** (*music*: *muspy.music.Music*) → *numpy.ndarray*  
Encode a Music object into piano-roll representation.

**Parameters** **music** (*muspy.Music* object) – Music object to encode.

**Returns** Encoded array in piano-roll representation.

**Return type** *ndarray* (*np.uint8*)

See also:

**`muspy.to_pianoroll_representation()`** Convert a Music object into piano-roll representation.

**class** `muspy.processors.PitchRepresentationProcessor` (*use\_hold\_state: bool = False*,  
*default\_velocity: int = 64*)

Pitch-based representation processor.

The pitch-based representation represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is  $T \times 1$ , where  $T$  is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

**use\_hold\_state**

Whether to use a special state for holds. Defaults to False.

**Type** `bool`

**default\_velocity**

Default velocity value to use when decoding. Defaults to 64.

**Type** `int`

**decode** (*array: numpy.ndarray*) → `muspy.music.Music`

Decode pitch-based representation into a Music object.

**Parameters** **array** (*ndarray*) – Array in pitch-based representation to decode. Cast to integer if not of integer type.

**Returns** Decoded Music object.

**Return type** `muspy.Music` object

**See also:**

**`muspy.from_pitch_representation()`** Return a Music object converted from pitch-based representation.

**encode** (*music: muspy.music.Music*) → `numpy.ndarray`

Encode a Music object into pitch-based representation.

**Parameters** **music** (*muspy.Music* object) – Music object to encode.

**Returns** Encoded array in pitch-based representation.

**Return type** `ndarray (np.uint8)`

**See also:**

**`muspy.to_pitch_representation()`** Convert a Music object into pitch-based representation.

### 7.10.8 muspy.schemas

JSON, YAML and MusicXML schemas.

This module provide functions for working with MusPy's JSON and YAML schemas and the MusicXML schema.

## Functions

- `get_json_schema_path`
- `get_musicxml_schema_path`
- `get_yaml_schema_path`

## Variables

- `DEFAULT_SCHEMA_VERSION`

`muspy.schemas.get_json_schema_path()` → str  
Return the path to the JSON schema.

`muspy.schemas.get_musicxml_schema_path()` → str  
Return the path to the MusicXML schema.

`muspy.schemas.get_yaml_schema_path()` → str  
Return the path to the YAML schema.

`muspy.schemas.validate_json(path: Union[str, pathlib.Path])`  
Validate a file against the JSON schema.

**Parameters** `path` (*str* or *Path*) – Path to the file to validate.

`muspy.schemas.validate_musicxml(path: Union[str, pathlib.Path])`  
Validate a file against the MusicXML schema.

**Parameters** `path` (*str* or *Path*) – Path to the file to validate.

`muspy.schemas.validate_yaml(path: Union[str, pathlib.Path])`  
Validate a file against the YAML schema.

**Parameters** `path` (*str* or *Path*) – Path to the file to validate.

## 7.10.9 muspy.visualization

Visualization tools.

This module provides functions for visualizing a Music object.

## Classes

- `ScorePlotter`

## Functions

- `show`
- `show_pianoroll`
- `show_score`

`muspy.visualization.show(music: Music, kind: str, **kwargs)`  
Show visualization.

**Parameters**



- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str*, {'piano-roll', 'score'}) – Target representation.

`muspy.visualization.show_pianoroll (music: Music, **kwargs)`

Show pianoroll visualization.

`muspy.visualization.show_score (music: Music, figsize: Optional[Tuple[float, float]] = None, clef: str = 'treble', clef_octave: Optional[int] = 0, note_spacing: Optional[int] = None, font_path: Union[str, pathlib.Path, None] = None, font_scale: Optional[float] = None) → muspy.visualization.score.ScorePlotter`

Show score visualization.

#### Parameters

- **music** (*muspy.Music*) – Music object to show.
- **figsize** ((*float*, *float*), *optional*) – Width and height in inches. Defaults to Matplotlib configuration.
- **clef** (*str*, {'treble', 'alto', 'bass'}) – Clef type. Defaults to a treble clef.
- **clef\_octave** (*int*) – Clef octave. Defaults to zero.
- **note\_spacing** (*int*, *optional*) – Spacing of notes. Defaults to 4.
- **font\_path** (*str* or *Path*, *optional*) – Path to the music font. Defaults to the path to the built-in Bravura font.
- **font\_scale** (*float*, *optional*) – Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

**Returns** A ScorePlotter object that handles the score.

**Return type** *muspy.ScorePlotter*

**class** `muspy.visualization.ScorePlotter` (*fig*: *matplotlib.figure.Figure*, *ax*: *matplotlib.axes.\_axes.Axes*, *resolution*: *int*, *note\_spacing*: *Optional[int] = None*, *font\_path*: *Union[str, pathlib.Path, None] = None*, *font\_scale*: *Optional[float] = None*)

A plotter that handles the score visualization.

**fig**

Figure object to plot the score on.

**Type** *matplotlib.figure.Figure*

**axes**

Axes object to plot the score on.

**Type** *matplotlib.axes.Axes*

**resolution**

Time steps per quarter note.

**Type** *int*

**note\_spacing**

Spacing of notes. Defaults to 4.

**Type** *int*, *optional*

**font\_path**

Path to the music font. Defaults to the path to the downloaded Bravura font.

**Type** `str` or `Path`, optional

**font\_scale**

Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

**Type** `float`, optional

**adjust\_fonts** (*scale: Optional[float] = None*)

Adjust the fonts.

**plot\_bar\_line** () → `matplotlib.lines.Line2D`

Plot a bar line.

**plot\_clef** (*kind='treble', octave=0*) → `matplotlib.text.Text`

Plot a clef.

**plot\_final\_bar\_line** () → `List[matplotlib.artist.Artist]`

Plot an ending bar line.

**plot\_key\_signature** (*root: int, mode: str*)

Plot a key signature. Supports only major and minor keys.

**plot\_note** (*time, duration, pitch*) → `Optional[Tuple[List[matplotlib.text.Text], List[matplotlib.patches.Arc]]]`

Plot a note.

**plot\_object** (*obj*)

Plot an object.

**plot\_staffs** (*start: Optional[float] = None, end: Optional[float] = None*) → `List[matplotlib.lines.Line2D]`

Plot the staffs.

**plot\_tempo** (*qpm*) → `List[matplotlib.artist.Artist]`

Plot a tempo as a metronome mark.

**plot\_time\_signature** (*numerator: int, denominator: int*) → `List[matplotlib.text.Text]`

Plot a time signature.

**set\_baseline** (*y*)

Set baseline position (y-coordinate of first staff line).

**update\_boundaries** (*left: Optional[float] = None, right: Optional[float] = None, bottom: Optional[float] = None, top: Optional[float] = None*)

Update boundaries.

### m

- `muspy`, [106](#)
- `muspy.datasets`, [157](#)
- `muspy.external`, [170](#)
- `muspy.inputs`, [171](#)
- `muspy.metrics`, [179](#)
- `muspy.outputs`, [185](#)
- `muspy.processors`, [191](#)
- `muspy.schemas`, [195](#)
- `muspy.visualization`, [196](#)



## A

ABCFolderDataset (class in muspy), 118  
 ABCFolderDataset (class in muspy.datasets), 158  
 adjust\_fonts() (muspy.ScorePlotter method), 157  
 adjust\_fonts() (muspy.visualization.ScorePlotter method), 198  
 adjust\_resolution() (in module muspy), 116  
 adjust\_resolution() (muspy.Music method), 144  
 adjust\_time() (in module muspy), 116  
 adjust\_time() (muspy.Base method), 107  
 adjust\_time() (muspy.Chord method), 111  
 adjust\_time() (muspy.Note method), 114  
 Annotation (class in muspy), 111  
 annotation (muspy.Annotation attribute), 47, 111  
 annotations (muspy.Music attribute), 19, 143  
 annotations (muspy.Track attribute), 26, 115  
 append() (in module muspy), 117  
 append() (muspy.ComplexBase method), 110  
 axes (in module muspy), 100  
 axes (muspy.ScorePlotter attribute), 156  
 axes (muspy.visualization.ScorePlotter attribute), 197

## B

Base (class in muspy), 106

## C

Chord (class in muspy), 111  
 chords (muspy.Track attribute), 26, 115  
 citation() (muspy.Dataset class method), 119  
 citation() (muspy.datasets.Dataset class method), 159  
 clip() (in module muspy), 117  
 clip() (muspy.Chord method), 112  
 clip() (muspy.Music method), 144  
 clip() (muspy.Note method), 114  
 clip() (muspy.Track method), 116  
 collection (muspy.Metadata attribute), 31, 113  
 ComplexBase (class in muspy), 110

convert() (muspy.datasets.FolderDataset method), 162  
 convert() (muspy.datasets.Music21Dataset method), 165  
 convert() (muspy.FolderDataset method), 122  
 convert() (muspy.Music21Dataset method), 124  
 converted\_dir (muspy.datasets.FolderDataset attribute), 162  
 converted\_dir (muspy.FolderDataset attribute), 122  
 converted\_exists() (muspy.datasets.FolderDataset method), 162  
 converted\_exists() (muspy.FolderDataset method), 122  
 copy() (muspy.Base method), 107  
 copyright (muspy.Metadata attribute), 31, 113  
 creators (muspy.Metadata attribute), 31, 113

## D

Dataset (class in muspy), 119  
 Dataset (class in muspy.datasets), 159  
 DatasetInfo (class in muspy), 121  
 DatasetInfo (class in muspy.datasets), 161  
 decode() (muspy.EventRepresentationProcessor method), 153  
 decode() (muspy.NoteRepresentationProcessor method), 152  
 decode() (muspy.PianoRollRepresentationProcessor method), 154  
 decode() (muspy.PitchRepresentationProcessor method), 155  
 decode() (muspy.processors.EventRepresentationProcessor method), 193  
 decode() (muspy.processors.NoteRepresentationProcessor method), 192  
 decode() (muspy.processors.PianoRollRepresentationProcessor method), 194  
 decode() (muspy.processors.PitchRepresentationProcessor method), 195  
 deepcopy() (muspy.Base method), 107

`default_velocity(muspy.EventRepresentationProcessor attribute)`, 95, 153  
`default_velocity(muspy.NoteRepresentationProcessor attribute)`, 97, 152  
`default_velocity(muspy.PianoRollRepresentationProcessor attribute)`, 92, 154  
`default_velocity(muspy.PitchRepresentationProcessor attribute)`, 91, 155  
`default_velocity(muspy.processors.EventRepresentationProcessor attribute)`, 193  
`default_velocity(muspy.processors.NoteRepresentationProcessor attribute)`, 192  
`default_velocity(muspy.processors.PianoRollRepresentationProcessor attribute)`, 194  
`default_velocity(muspy.processors.PitchRepresentationProcessor attribute)`, 195  
`denominator(muspy.TimeSignature attribute)`, 41, 115  
`downbeats(muspy.Music attribute)`, 19, 143  
`download()(muspy.datasets.HymnalDataset method)`, 163  
`download()(muspy.datasets.HymnalTuneDataset method)`, 163  
`download()(muspy.datasets.RemoteDataset method)`, 167  
`download()(muspy.HymnalDataset method)`, 123  
`download()(muspy.HymnalTuneDataset method)`, 123  
`download()(muspy.RemoteDataset method)`, 127  
`download_and_extract()(muspy.datasets.RemoteDataset method)`, 168  
`download_and_extract()(muspy.RemoteDataset method)`, 127  
`download_bravura_font()(in module muspy)`, 129  
`download_bravura_font()(in module muspy.external)`, 170  
`download_musescore_soundfont()(in module muspy)`, 129  
`download_musescore_soundfont()(in module muspy.external)`, 170  
`drum_in_pattern_rate()(in module muspy)`, 137  
`drum_in_pattern_rate()(in module muspy.metrics)`, 180  
`drum_pattern_consistency()(in module muspy)`, 138  
`drum_pattern_consistency()(in module muspy.metrics)`, 180  
`dtype(muspy.NoteRepresentationProcessor attribute)`, 97, 152  
`dtype(muspy.processors.NoteRepresentationProcessor attribute)`, 192  
`duration(muspy.Chord attribute)`, 54, 111  
`duration(muspy.Note attribute)`, 50, 113  
`empty_beat_rate()(in module muspy)`, 138  
`empty_beat_rate()(in module muspy.metrics)`, 180  
`empty_measure_rate()(in module muspy)`, 138  
`empty_measure_rate()(in module muspy.metrics)`, 181  
`encode()(muspy.EventRepresentationProcessor method)`, 154  
`encode()(muspy.NoteRepresentationProcessor method)`, 152  
`encode()(muspy.PianoRollRepresentationProcessor method)`, 154  
`encode()(muspy.PitchRepresentationProcessor method)`, 155  
`encode()(muspy.processors.EventRepresentationProcessor method)`, 194  
`encode()(muspy.processors.NoteRepresentationProcessor method)`, 192  
`encode()(muspy.processors.PianoRollRepresentationProcessor method)`, 194  
`encode()(muspy.processors.PitchRepresentationProcessor method)`, 195  
`encode_velocity(muspy.EventRepresentationProcessor attribute)`, 95, 153  
`encode_velocity(muspy.NoteRepresentationProcessor attribute)`, 97, 152  
`encode_velocity(muspy.PianoRollRepresentationProcessor attribute)`, 92, 154  
`encode_velocity(muspy.processors.EventRepresentationProcessor attribute)`, 193  
`encode_velocity(muspy.processors.NoteRepresentationProcessor attribute)`, 192  
`encode_velocity(muspy.processors.PianoRollRepresentationProcessor attribute)`, 194  
`end(muspy.Chord attribute)`, 112  
`end(muspy.Note attribute)`, 114  
`EssenFolkSongDatabase(class in muspy)`, 121  
`EssenFolkSongDatabase(class in muspy.datasets)`, 161  
`EventRepresentationProcessor(class in muspy)`, 153  
`EventRepresentationProcessor(class in muspy.processors)`, 192  
`exists()(muspy.datasets.FolderDataset method)`, 162  
`exists()(muspy.datasets.RemoteDataset method)`, 168  
`exists()(muspy.FolderDataset method)`, 122  
`exists()(muspy.RemoteDataset method)`, 127  
`extend()(muspy.ComplexBase method)`, 110  
`extract()(muspy.datasets.RemoteDataset method)`, 168  
`extract()(muspy.RemoteDataset method)`, 127

## F

[fifths \(muspy.KeySignature attribute\)](#), 38, 112  
[fig \(in module muspy\)](#), 100  
[fig \(muspy.ScorePlotter attribute\)](#), 156  
[fig \(muspy.visualization.ScorePlotter attribute\)](#), 197  
[FolderDataset \(class in muspy\)](#), 121  
[FolderDataset \(class in muspy.datasets\)](#), 161  
[font\\_path \(in module muspy\)](#), 100  
[font\\_path \(muspy.ScorePlotter attribute\)](#), 157  
[font\\_path \(muspy.visualization.ScorePlotter attribute\)](#), 197  
[font\\_scale \(in module muspy\)](#), 100  
[font\\_scale \(muspy.ScorePlotter attribute\)](#), 157  
[font\\_scale \(muspy.visualization.ScorePlotter attribute\)](#), 198  
[force\\_velocity\\_event \(muspy.EventRepresentationProcessor attribute\)](#), 95, 153  
[force\\_velocity\\_event \(muspy.processors.EventRepresentationProcessor attribute\)](#), 193  
[from\\_dict \(\) \(muspy.Base class method\)](#), 107  
[from\\_event\\_representation \(\) \(in module muspy\)](#), 130  
[from\\_event\\_representation \(\) \(in module muspy.inputs\)](#), 171  
[from\\_mido \(\) \(in module muspy\)](#), 131  
[from\\_mido \(\) \(in module muspy.inputs\)](#), 172  
[from\\_music21 \(\) \(in module muspy\)](#), 131  
[from\\_music21 \(\) \(in module muspy.inputs\)](#), 173  
[from\\_music21\\_opus \(\) \(in module muspy\)](#), 131  
[from\\_music21\\_opus \(\) \(in module muspy.inputs\)](#), 173  
[from\\_music21\\_part \(\) \(in module muspy\)](#), 131  
[from\\_music21\\_part \(\) \(in module muspy.inputs\)](#), 173  
[from\\_music21\\_score \(\) \(in module muspy\)](#), 132  
[from\\_music21\\_score \(\) \(in module muspy.inputs\)](#), 173  
[from\\_note\\_representation \(\) \(in module muspy\)](#), 132  
[from\\_note\\_representation \(\) \(in module muspy.inputs\)](#), 174  
[from\\_object \(\) \(in module muspy\)](#), 132  
[from\\_object \(\) \(in module muspy.inputs\)](#), 174  
[from\\_pianoroll\\_representation \(\) \(in module muspy\)](#), 133  
[from\\_pianoroll\\_representation \(\) \(in module muspy.inputs\)](#), 174  
[from\\_pitch\\_representation \(\) \(in module muspy\)](#), 133  
[from\\_pitch\\_representation \(\) \(in module muspy.inputs\)](#), 175  
[from\\_pretty\\_midi \(\) \(in module muspy\)](#), 134

[from\\_pretty\\_midi \(\) \(in module muspy.inputs\)](#), 176  
[from\\_pypianoroll \(\) \(in module muspy\)](#), 134  
[from\\_pypianoroll \(\) \(in module muspy.inputs\)](#), 176  
[from\\_pypianoroll\\_track \(\) \(in module muspy\)](#), 134  
[from\\_pypianoroll\\_track \(\) \(in module muspy.inputs\)](#), 176  
[from\\_representation \(\) \(in module muspy\)](#), 134  
[from\\_representation \(\) \(in module muspy.inputs\)](#), 176

## G

[get\\_bravura\\_font\\_dir \(\) \(in module muspy\)](#), 129  
[get\\_bravura\\_font\\_dir \(\) \(in module muspy.external\)](#), 170  
[get\\_bravura\\_font\\_path \(\) \(in module muspy\)](#), 129  
[get\\_bravura\\_font\\_path \(\) \(in module muspy.external\)](#), 170  
[get\\_dataset \(\) \(in module muspy\)](#), 129  
[get\\_dataset \(\) \(in module muspy.datasets\)](#), 170  
[get\\_end\\_time \(\) \(in module muspy\)](#), 117  
[get\\_end\\_time \(\) \(muspy.Music method\)](#), 144  
[get\\_end\\_time \(\) \(muspy.Track method\)](#), 116  
[get\\_json\\_schema\\_path \(\) \(in module muspy\)](#), 155  
[get\\_json\\_schema\\_path \(\) \(in module muspy.schemas\)](#), 196  
[get\\_musescore\\_soundfont\\_dir \(\) \(in module muspy\)](#), 129  
[get\\_musescore\\_soundfont\\_dir \(\) \(in module muspy.external\)](#), 170  
[get\\_musescore\\_soundfont\\_path \(\) \(in module muspy\)](#), 129  
[get\\_musescore\\_soundfont\\_path \(\) \(in module muspy.external\)](#), 170  
[get\\_musicxml\\_schema\\_path \(\) \(in module muspy\)](#), 155  
[get\\_musicxml\\_schema\\_path \(\) \(in module muspy.schemas\)](#), 196  
[get\\_real\\_end\\_time \(\) \(in module muspy\)](#), 117  
[get\\_real\\_end\\_time \(\) \(muspy.Music method\)](#), 144  
[get\\_yaml\\_schema\\_path \(\) \(in module muspy\)](#), 155  
[get\\_yaml\\_schema\\_path \(\) \(in module muspy.schemas\)](#), 196  
[groove\\_consistency \(\) \(in module muspy\)](#), 139  
[groove\\_consistency \(\) \(in module muspy.metrics\)](#), 181  
[group \(muspy.Annotation attribute\)](#), 47, 111

## H

[HaydnOp20Dataset \(class in muspy\)](#), 122  
[HaydnOp20Dataset \(class in muspy.datasets\)](#), 163  
[HymnalDataset \(class in muspy\)](#), 123  
[HymnalDataset \(class in muspy.datasets\)](#), 163



HymnalTuneDataset (class in muspy), 123  
 HymnalTuneDataset (class in muspy.datasets), 163

## I

info() (muspy.Dataset class method), 119  
 info() (muspy.datasets.Dataset class method), 159  
 is\_drum (muspy.Track attribute), 26, 115  
 is\_valid() (muspy.Base method), 108  
 is\_valid\_type() (muspy.Base method), 108

## J

JSBChoralesDataset (class in muspy), 123  
 JSBChoralesDataset (class in muspy.datasets), 163

## K

key\_signatures (muspy.Music attribute), 19, 143  
 KeySignature (class in muspy), 112  
 kind (muspy.datasets.MusicDataset attribute), 165  
 kind (muspy.datasets.RemoteMusicDataset attribute), 169  
 kind (muspy.MusicDataset attribute), 73, 125  
 kind (muspy.RemoteMusicDataset attribute), 83, 128

## L

LakhMIDIAlignedDataset (class in muspy), 123  
 LakhMIDIAlignedDataset (class in muspy.datasets), 164  
 LakhMIDIAlignedDataset (class in muspy), 123  
 LakhMIDIAlignedDataset (class in muspy.datasets), 164  
 LakhMIDIMatchedDataset (class in muspy), 124  
 LakhMIDIMatchedDataset (class in muspy.datasets), 164  
 list\_datasets() (in module muspy), 129  
 list\_datasets() (in module muspy.datasets), 170  
 load() (in module muspy), 135  
 load() (in module muspy.inputs), 177  
 load() (muspy.datasets.FolderDataset method), 162  
 load() (muspy.FolderDataset method), 122  
 load\_json() (in module muspy), 135  
 load\_json() (in module muspy.inputs), 177  
 load\_yaml() (in module muspy), 135  
 load\_yaml() (in module muspy.inputs), 177  
 Lyric (class in muspy), 112  
 lyric (muspy.Lyric attribute), 44, 113  
 lyrics (muspy.Music attribute), 19, 143  
 lyrics (muspy.Track attribute), 26, 115

## M

MAESTRODatasetV1 (class in muspy), 124  
 MAESTRODatasetV1 (class in muspy.datasets), 164  
 MAESTRODatasetV2 (class in muspy), 124  
 MAESTRODatasetV2 (class in muspy.datasets), 164  
 max\_time\_shift (muspy.EventRepresentationProcessor on\_the\_fly() attribute), 95, 153

max\_time\_shift (muspy.processors.EventRepresentationProcessor attribute), 193

Metadata (class in muspy), 113  
 metadata (muspy.Music attribute), 19, 143  
 MIDIError, 129, 171  
 mode (muspy.KeySignature attribute), 38, 112  
 Music (class in muspy), 142  
 Music21Dataset (class in muspy), 124  
 Music21Dataset (class in muspy.datasets), 165  
 MusicDataset (class in muspy), 125  
 MusicDataset (class in muspy.datasets), 165  
 MusicNetDataset (class in muspy), 125  
 MusicNetDataset (class in muspy.datasets), 165  
 MusicXMLError, 130, 171  
 muspy (module), 106  
 muspy.datasets (module), 157  
 muspy.external (module), 170  
 muspy.inputs (module), 171  
 muspy.metrics (module), 179  
 muspy.outputs (module), 185  
 muspy.processors (module), 191  
 muspy.schemas (module), 195  
 muspy.visualization (module), 196

## N

n\_pitch\_classes\_used() (in module muspy), 139  
 n\_pitch\_classes\_used() (in module muspy.metrics), 182  
 n\_pitches\_used() (in module muspy), 140  
 n\_pitches\_used() (in module muspy.metrics), 182  
 name (muspy.Track attribute), 26, 115  
 NESMusicDatabase (class in muspy), 125  
 NESMusicDatabase (class in muspy.datasets), 166  
 Note (class in muspy), 113  
 note\_spacing (in module muspy), 100  
 note\_spacing (muspy.ScorePlotter attribute), 157  
 note\_spacing (muspy.visualization.ScorePlotter attribute), 197  
 NoteRepresentationProcessor (class in muspy), 152  
 NoteRepresentationProcessor (class in muspy.processors), 192  
 notes (muspy.Track attribute), 26, 115  
 NottinghamDatabase (class in muspy), 125  
 NottinghamDatabase (class in muspy.datasets), 166  
 numerator (muspy.TimeSignature attribute), 41, 115

## O

on\_the\_fly() (muspy.ABCFolderDataset method), 118  
 on\_the\_fly() (muspy.datasets.ABCFolderDataset method), 159  
 on\_the\_fly() (muspy.datasets.FolderDataset method), 162



`on_the_fly()` (*muspy.FolderDataset* method), 122

## P

*PianoRollRepresentationProcessor* (class in *muspy*), 154

*PianoRollRepresentationProcessor* (class in *muspy.processors*), 194

`pitch` (*muspy.Note* attribute), 50, 113

`pitch_class_entropy()` (in module *muspy*), 140

`pitch_class_entropy()` (in module *muspy.metrics*), 182

`pitch_entropy()` (in module *muspy*), 140

`pitch_entropy()` (in module *muspy.metrics*), 183

`pitch_in_scale_rate()` (in module *muspy*), 141

`pitch_in_scale_rate()` (in module *muspy.metrics*), 183

`pitch_range()` (in module *muspy*), 141

`pitch_range()` (in module *muspy.metrics*), 183

`pitch_str` (*muspy.Note* attribute), 50, 114

`pitches` (*muspy.Chord* attribute), 54, 111

`pitches_str` (*muspy.Chord* attribute), 54, 111

*PitchRepresentationProcessor* (class in *muspy*), 155

*PitchRepresentationProcessor* (class in *muspy.processors*), 195

`plot_bar_line()` (*muspy.ScorePlotter* method), 157

`plot_bar_line()` (*muspy.visualization.ScorePlotter* method), 198

`plot_clef()` (*muspy.ScorePlotter* method), 157

`plot_clef()` (*muspy.visualization.ScorePlotter* method), 198

`plot_final_bar_line()` (*muspy.ScorePlotter* method), 157

`plot_final_bar_line()` (*muspy.visualization.ScorePlotter* method), 198

`plot_key_signature()` (*muspy.ScorePlotter* method), 157

`plot_key_signature()` (*muspy.visualization.ScorePlotter* method), 198

`plot_note()` (*muspy.ScorePlotter* method), 157

`plot_note()` (*muspy.visualization.ScorePlotter* method), 198

`plot_object()` (*muspy.ScorePlotter* method), 157

`plot_object()` (*muspy.visualization.ScorePlotter* method), 198

`plot_staffs()` (*muspy.ScorePlotter* method), 157

`plot_staffs()` (*muspy.visualization.ScorePlotter* method), 198

`plot_tempo()` (*muspy.ScorePlotter* method), 157

`plot_tempo()` (*muspy.visualization.ScorePlotter* method), 198

`plot_time_signature()` (*muspy.ScorePlotter* method), 157

`plot_time_signature()` (*muspy.visualization.ScorePlotter* method), 198

`polyphony()` (in module *muspy*), 141

`polyphony()` (in module *muspy.metrics*), 184

`polyphony_rate()` (in module *muspy*), 141

`polyphony_rate()` (in module *muspy.metrics*), 184

`pretty_str()` (*muspy.Base* method), 108

`print()` (*muspy.Base* method), 109

`program` (*muspy.Track* attribute), 26, 115

## Q

`qpm` (*muspy.Tempo* attribute), 34, 115

## R

`read()` (in module *muspy*), 136

`read()` (in module *muspy.inputs*), 178

`read()` (*muspy.ABCFolderDataset* method), 118

`read()` (*muspy.datasets.ABCFolderDataset* method), 159

`read()` (*muspy.datasets.FolderDataset* method), 163

`read()` (*muspy.datasets.HaydnOp20Dataset* method), 163

`read()` (*muspy.datasets.HymnalDataset* method), 163

`read()` (*muspy.datasets.HymnalTuneDataset* method), 163

`read()` (*muspy.datasets.JSBChoralesDataset* method), 164

`read()` (*muspy.datasets.LakhMIDIALignedDataset* method), 164

`read()` (*muspy.datasets.LakhMIDIataset* method), 164

`read()` (*muspy.datasets.LakhMIDIMatchedDataset* method), 164

`read()` (*muspy.datasets.MAESTRODatasetV1* method), 164

`read()` (*muspy.datasets.MAESTRODatasetV2* method), 165

`read()` (*muspy.datasets.MusicNetDataset* method), 166

`read()` (*muspy.datasets.NESMusicDatabase* method), 166

`read()` (*muspy.datasets.RemoteFolderDataset* method), 169

`read()` (*muspy.datasets.WikifoniaDataset* method), 170

`read()` (*muspy.FolderDataset* method), 122

`read()` (*muspy.HaydnOp20Dataset* method), 123

`read()` (*muspy.HymnalDataset* method), 123

`read()` (*muspy.HymnalTuneDataset* method), 123

`read()` (*muspy.JSBChoralesDataset* method), 123

`read()` (*muspy.LakhMIDIALignedDataset* method), 123

`read()` (*muspy.LakhMIDIataset* method), 124

- [read\(\)](#) (*muspy.LakhMIDIMatchedDataset* method), 124  
[read\(\)](#) (*muspy.MAESTRODatasetV1* method), 124  
[read\(\)](#) (*muspy.MAESTRODatasetV2* method), 124  
[read\(\)](#) (*muspy.MusicNetDataset* method), 125  
[read\(\)](#) (*muspy.NESMusicDatabase* method), 125  
[read\(\)](#) (*muspy.RemoteFolderDataset* method), 128  
[read\(\)](#) (*muspy.WikifoniaDataset* method), 129  
[read\\_abc\(\)](#) (in module *muspy*), 136  
[read\\_abc\(\)](#) (in module *muspy.inputs*), 178  
[read\\_abc\\_string\(\)](#) (in module *muspy*), 136  
[read\\_abc\\_string\(\)](#) (in module *muspy.inputs*), 178  
[read\\_midi\(\)](#) (in module *muspy*), 137  
[read\\_midi\(\)](#) (in module *muspy.inputs*), 178  
[read\\_musicxml\(\)](#) (in module *muspy*), 137  
[read\\_musicxml\(\)](#) (in module *muspy.inputs*), 179  
[RemoteABCFolderDataset](#) (class in *muspy*), 125  
[RemoteABCFolderDataset](#) (class in *muspy.datasets*), 166  
[RemoteDataset](#) (class in *muspy*), 126  
[RemoteDataset](#) (class in *muspy.datasets*), 166  
[RemoteFolderDataset](#) (class in *muspy*), 128  
[RemoteFolderDataset](#) (class in *muspy.datasets*), 168  
[RemoteMusicDataset](#) (class in *muspy*), 128  
[RemoteMusicDataset](#) (class in *muspy.datasets*), 169  
[remove\\_duplicate\(\)](#) (in module *muspy*), 118  
[remove\\_duplicate\(\)](#) (*muspy.ComplexBase* method), 110  
[remove\\_invalid\(\)](#) (*muspy.ComplexBase* method), 110  
[resolution](#) (in module *muspy*), 100  
[resolution](#) (*muspy.Music* attribute), 19, 143  
[resolution](#) (*muspy.ScorePlotter* attribute), 156  
[resolution](#) (*muspy.visualization.ScorePlotter* attribute), 197  
[root](#) (*muspy.datasets.FolderDataset* attribute), 161  
[root](#) (*muspy.datasets.MusicDataset* attribute), 165  
[root](#) (*muspy.datasets.RemoteDataset* attribute), 166  
[root](#) (*muspy.datasets.RemoteFolderDataset* attribute), 168  
[root](#) (*muspy.datasets.RemoteMusicDataset* attribute), 169  
[root](#) (*muspy.FolderDataset* attribute), 70, 121  
[root](#) (*muspy.KeySignature* attribute), 37, 112  
[root](#) (*muspy.MusicDataset* attribute), 73, 125  
[root](#) (*muspy.RemoteDataset* attribute), 66, 126  
[root](#) (*muspy.RemoteFolderDataset* attribute), 79, 128  
[root](#) (*muspy.RemoteMusicDataset* attribute), 83, 128  
[root\\_str](#) (*muspy.KeySignature* attribute), 38, 112
- S**
- [save\(\)](#) (in module *muspy*), 146  
[save\(\)](#) (in module *muspy.outputs*), 186  
[save\(\)](#) (*muspy.Dataset* method), 119  
[save\(\)](#) (*muspy.datasets.Dataset* method), 159  
[save\(\)](#) (*muspy.Music* method), 144  
[save\\_json\(\)](#) (in module *muspy*), 147  
[save\\_json\(\)](#) (in module *muspy.outputs*), 186  
[save\\_json\(\)](#) (*muspy.Music* method), 144  
[save\\_yaml\(\)](#) (in module *muspy*), 147  
[save\\_yaml\(\)](#) (in module *muspy.outputs*), 187  
[save\\_yaml\(\)](#) (*muspy.Music* method), 144  
[scale\\_consistency\(\)](#) (in module *muspy*), 142  
[scale\\_consistency\(\)](#) (in module *muspy.metrics*), 184  
[schema\\_version](#) (*muspy.Metadata* attribute), 31, 113  
[ScorePlotter](#) (class in *muspy*), 156  
[ScorePlotter](#) (class in *muspy.visualization*), 197  
[set\\_baseline\(\)](#) (*muspy.ScorePlotter* method), 157  
[set\\_baseline\(\)](#) (*muspy.visualization.ScorePlotter* method), 198  
[show\(\)](#) (in module *muspy*), 156  
[show\(\)](#) (in module *muspy.visualization*), 196  
[show\(\)](#) (*muspy.Music* method), 145  
[show\\_pianoroll\(\)](#) (in module *muspy*), 156  
[show\\_pianoroll\(\)](#) (in module *muspy.visualization*), 197  
[show\\_pianoroll\(\)](#) (*muspy.Music* method), 145  
[show\\_score\(\)](#) (in module *muspy*), 156  
[show\\_score\(\)](#) (in module *muspy.visualization*), 197  
[show\\_score\(\)](#) (*muspy.Music* method), 145  
[sort\(\)](#) (in module *muspy*), 118  
[sort\(\)](#) (*muspy.ComplexBase* method), 110  
[source\\_exists\(\)](#) (*muspy.datasets.RemoteDataset* method), 168  
[source\\_exists\(\)](#) (*muspy.RemoteDataset* method), 128  
[source\\_filename](#) (*muspy.Metadata* attribute), 31, 113  
[source\\_format](#) (*muspy.Metadata* attribute), 31, 113  
[split\(\)](#) (*muspy.Dataset* method), 119  
[split\(\)](#) (*muspy.datasets.Dataset* method), 159  
[start](#) (*muspy.Chord* attribute), 112  
[start](#) (*muspy.Note* attribute), 114  
[synthesize\(\)](#) (in module *muspy*), 150  
[synthesize\(\)](#) (in module *muspy.outputs*), 190  
[synthesize\(\)](#) (*muspy.Music* method), 145
- T**
- [Tempo](#) (class in *muspy*), 114  
[tempos](#) (*muspy.Music* attribute), 19, 143  
[time](#) (*muspy.Annotation* attribute), 47, 111  
[time](#) (*muspy.Chord* attribute), 53, 111  
[time](#) (*muspy.KeySignature* attribute), 37, 112  
[time](#) (*muspy.Lyric* attribute), 44, 112  
[time](#) (*muspy.Note* attribute), 50, 113  
[time](#) (*muspy.Tempo* attribute), 34, 114

- time (*muspy.TimeSignature* attribute), 41, 115  
 time\_signatures (*muspy.Music* attribute), 19, 143  
 TimeSignature (class in *muspy*), 115  
 title (*muspy.Metadata* attribute), 31, 113  
 to\_event\_representation() (in module *muspy*), 148  
 to\_event\_representation() (in module *muspy.outputs*), 187  
 to\_event\_representation() (*muspy.Music* method), 145  
 to\_mido() (in module *muspy*), 148  
 to\_mido() (in module *muspy.outputs*), 188  
 to\_mido() (*muspy.Music* method), 145  
 to\_music21() (in module *muspy*), 148  
 to\_music21() (in module *muspy.outputs*), 188  
 to\_music21() (*muspy.Music* method), 145  
 to\_note\_representation() (in module *muspy*), 149  
 to\_note\_representation() (in module *muspy.outputs*), 188  
 to\_note\_representation() (*muspy.Music* method), 145  
 to\_object() (in module *muspy*), 149  
 to\_object() (in module *muspy.outputs*), 188  
 to\_object() (*muspy.Music* method), 145  
 to\_ordered\_dict() (in module *muspy*), 118  
 to\_ordered\_dict() (*muspy.Base* method), 109  
 to\_pianoroll\_representation() (in module *muspy*), 149  
 to\_pianoroll\_representation() (in module *muspy.outputs*), 189  
 to\_pianoroll\_representation() (*muspy.Music* method), 145  
 to\_pitch\_representation() (in module *muspy*), 149  
 to\_pitch\_representation() (in module *muspy.outputs*), 189  
 to\_pitch\_representation() (*muspy.Music* method), 145  
 to\_pretty\_midi() (in module *muspy*), 150  
 to\_pretty\_midi() (in module *muspy.outputs*), 189  
 to\_pretty\_midi() (*muspy.Music* method), 145  
 to\_pypianoroll() (in module *muspy*), 150  
 to\_pypianoroll() (in module *muspy.outputs*), 189  
 to\_pypianoroll() (*muspy.Music* method), 145  
 to\_pytorch\_dataset() (*muspy.Dataset* method), 120  
 to\_pytorch\_dataset() (*muspy.datasets.Dataset* method), 160  
 to\_representation() (in module *muspy*), 150  
 to\_representation() (in module *muspy.outputs*), 190  
 to\_representation() (*muspy.Music* method), 146  
 to\_tensorflow\_dataset() (*muspy.Dataset* method), 120  
 to\_tensorflow\_dataset() (*muspy.datasets.Dataset* method), 160  
 Track (class in *muspy*), 115  
 tracks (*muspy.Music* attribute), 19, 143  
 transpose() (in module *muspy*), 118  
 transpose() (*muspy.Chord* method), 112  
 transpose() (*muspy.Music* method), 146  
 transpose() (*muspy.Note* method), 114  
 transpose() (*muspy.Track* method), 116
- ## U
- update\_boundaries() (*muspy.ScorePlotter* method), 157  
 update\_boundaries() (*muspy.visualization.ScorePlotter* method), 198  
 use\_converted() (*muspy.datasets.FolderDataset* method), 163  
 use\_converted() (*muspy.FolderDataset* method), 122  
 use\_end\_of\_sequence\_event (*muspy.EventRepresentationProcessor* attribute), 95, 153  
 use\_end\_of\_sequence\_event (*muspy.processors.EventRepresentationProcessor* attribute), 193  
 use\_hold\_state (*muspy.PitchRepresentationProcessor* attribute), 91, 155  
 use\_hold\_state (*muspy.processors.PitchRepresentationProcessor* attribute), 195  
 use\_single\_note\_off\_event (*muspy.EventRepresentationProcessor* attribute), 95, 153  
 use\_single\_note\_off\_event (*muspy.processors.EventRepresentationProcessor* attribute), 193  
 use\_start\_end (*muspy.NoteRepresentationProcessor* attribute), 97, 152  
 use\_start\_end (*muspy.processors.NoteRepresentationProcessor* attribute), 192
- ## V
- validate() (*muspy.Base* method), 109  
 validate\_json() (in module *muspy*), 155  
 validate\_json() (in module *muspy.schemas*), 196  
 validate\_musicxml() (in module *muspy*), 156  
 validate\_musicxml() (in module *muspy.schemas*), 196  
 validate\_type() (*muspy.Base* method), 109  
 validate\_yaml() (in module *muspy*), 156  
 validate\_yaml() (in module *muspy.schemas*), 196  
 velocity (*muspy.Chord* attribute), 54, 111  
 velocity (*muspy.Note* attribute), 50, 114

`velocity_bins` (*muspy.EventRepresentationProcessor*  
*attribute*), 95, 153  
`velocity_bins` (*muspy.processors.EventRepresentationProcessor*  
*attribute*), 193

## W

`WikifoniaDataset` (*class in muspy*), 129  
`WikifoniaDataset` (*class in muspy.datasets*), 169  
`write()` (*in module muspy*), 151  
`write()` (*in module muspy.outputs*), 190  
`write()` (*muspy.Music method*), 146  
`write_abc()` (*in module muspy*), 151  
`write_abc()` (*in module muspy.outputs*), 190  
`write_abc()` (*muspy.Music method*), 146  
`write_audio()` (*in module muspy*), 151  
`write_audio()` (*in module muspy.outputs*), 190  
`write_audio()` (*muspy.Music method*), 146  
`write_midi()` (*in module muspy*), 151  
`write_midi()` (*in module muspy.outputs*), 191  
`write_midi()` (*muspy.Music method*), 146  
`write_musicxml()` (*in module muspy*), 151  
`write_musicxml()` (*in module muspy.outputs*), 191  
`write_musicxml()` (*muspy.Music method*), 146