Internal objects

These objects are mainly used by pyo itself, inside other objects.

Dummy

class **Dummy**(*objs_list*) [source]

Dummy object used to perform arithmetics on PyoObject.

The user should never instantiate an object of this class.

Parent: `PyoObject`

Args: objs_list: list of audio Stream objects

List of Stream objects return by the PyoObject hidden method getBaseObjects().

Note: Multiplication, addition, division and substraction don't changed the PyoObject on which the operation is performed. A dummy object is created, which is just a copy of the audio Streams of the object, and the operation is applied on the Dummy, leaving the original object unchanged. This lets the user performs multiple different arithmetic operations on an object without conficts. Here, *b* is a Dummy object with *a* as its input with a *mul* attribute of 0.5. attribute:

```
>>> a = Sine()
>>> b = a * .5
>>> print(a)
<pyolib.input.Sine object at 0x11fd610>
>>> print(b)
<pyolib._core.Dummy object at 0x11fd710>
```

```
>>> s = Server().boot()
>>> s.start()
>>> m = Metro(time=0.25).play()
>>> p = TrigChoice(m, choice=[midiToHz(n) for n in [60,62,65,67,69]])
>>> a = SineLoop(p, feedback=.05, mul=.1).mix(2).out()
>>> b = SineLoop(p*1.253, feedback=.05, mul=.06).mix(2).out()
>>> c = SineLoop(p*1.497, feedback=.05, mul=.03).mix(2).out()
```

InputFader

class **InputFader**(*input*) [source]

Audio streams crossfader.

Args: input: PyoObject

Input signal.

Note: The setInput method, available to object with *input* attribute, uses an InputFader object internally to perform crossfade between the old and the new audio input assigned to the object.

```
>>> s = Server().boot()
>>> s.start()
>>> a = SineLoop([449,450], feedback=0.05, mul=.2)
>>> b = SineLoop([650,651], feedback=0.05, mul=.2)
>>> c = InputFader(a).out()
>>> # to created a crossfade, assign a new audio input:
>>> c.setInput(b, fadetime=5)
```

input

PyoObject. Input signal.

setInput(*x*, *fadetime=0.05*) [source]

Replace the *input* attribute.

Args: x: PyoObject

New signal to process.

fadetime: float, optional

Crossfade time between old and new input. Defaults to 0.05.

Mix

class **Mix**(*input*, *voices=1*, *mul=1*, *add=0*) [source]

Mix audio streams to arbitrary number of streams.

Mix the object's audio streams as *input* argument into *voices* streams.

Parent: PyoObject

Args: input: PyoObject or list of PyoObjects

Input signal(s) to mix the streams.

voices: int, optional

Number of streams of the Mix object. If more than 1, input object's streams are alternated and added into Mix object's streams. Defaults to 1.

Note: The mix method of PyoObject creates and returns a new Mix object with mixed streams of the object that called the method. User don't have to instantiate this class directly. These two calls are identical:

```
>>> b = a.mix()
>>> b = Mix(a)
```

```
>>> s = Server().boot()
>>> s.start()
>>> a = Sine([random.uniform(400,600) for i in range(50)], mul=.02)
>>> b = Mix(a, voices=2).out()
>>> print(len(a))
50
>>> print(len(b))
1
```

VarPort

class **VarPort**(*value*, *time=0.025*, *init=0.0*, *function=None*, *arg=None*, *mul=1*, *add=0*) [source]

Convert numeric value to PyoObject signal with portamento.

When *value* attribute is changed, a smoothed ramp is applied from the current value to the new value. If a callback is provided as *function* argument, it will be called at the end of the line.

Parent: PyoObject

Args: value: float

Numerical value to convert.

time: float, optional

Ramp time, in seconds, to reach the new value. Defaults to 0.025.

init: float, optional

Initial value of the internal memory. Defaults to 0.

function: Python callable, optional

If provided, it will be called at the end of the line. Defaults to None.

arg: any Python object, optional

Optional argument sent to the function called at the end of the line. Defaults to None.

Note: The out() method is bypassed. VarPort's signal can not be sent to audio outs.

```
>>> s = Server().boot()
>>> s.start()
>>> def callback(arg):
...     print("end of line")
...     print(arg)
...
>>> fr = VarPort(value=500, time=2, init=250, function=callback, arg="YEP!")
>>> a = SineLoop(freq=[fr,fr*1.01], feedback=0.05, mul=.2).out()
```

function

Python callable. Function to be called.

`setFunction(x)` [source]

Replace the *function* attribute.

Args: x: Python function
new *function* attribute.

`setTime(x)` [source]

Changes the ramp time of the object.

Args: x: float
New ramp time.

`setValue(x)` [source]

Changes the value of the signal stream.

Args: x: float
Numerical value to convert.

`time`

float. Ramp time.

`value`

float. Numerical value to convert.

Stream

class **Stream**

Audio stream objects. For internal use only.

A Stream object must never be instantiated by the user.

A Stream is a mono buffer of audio samples. It is used to pass audio between objects and the server. A PyoObject can manage many streams if, for example, a list is given to a parameter.

A Sine object with only one stream:

```
>>> a = Sine(freq=1000)
>>> print len(a)
1
```

A Sine object with four streams:

```
>>> a = Sine(freq=[250,500,750,100])
>>> print len(a)
```

```
4
```

The first stream of this object contains the samples from the 250Hz waveform. The second stream contains the samples from the 500Hz waveform, and so on.

User can call a specific stream of an object by giving the position of the stream between brackets, beginning at 0. To retrieve only the third stream of our object:

```
>>> a[2].out()
```

The method getStreamObject() can be called on a Stream object to retrieve the XXX_base object associated with this Stream. This method can be used by developers who are debugging their programs!