Event Sequencing

Set of objects that call Python functions from triggers or number counts. Useful for event sequencing.

CallAfter

class **CallAfter**(*function*, *time=1*, *arg=None*) [source]

Calls a Python function after a given time.

Parent: PyoObject

Args: function: Python function

Python callable execute after *time* seconds.

time: float, optional

Time, in seconds, before the call. Default to 1.

arg: any Python object, optional

Argument sent to the called function. Default to None.

Note: The out() method is bypassed. CallAfter doesn't return signal.
CallAfter has no *mul* and *add* attributes.

If *arg* is None, the function must be defined without argument:

```
>>> def tocall():
>>>     # function's body
```

If *arg* is not None, the function must be defined with one argument:

```
>>> def tocall(arg):
>>>     print(arg)
```

The object is not deleted after the call. The user must delete it himself.

```
>>> s = Server().boot()
>>> s.start()
>>> # Start an oscillator with a frequency of 250 Hz
>>> syn = SineLoop(freq=[250,251], feedback=.07, mul=.2).out()
>>> def callback(arg):
...     # Change the oscillator's frequency to 300 Hz after 2 seconds
...     syn.freq = arg
>>> a = CallAfter(callback, 2, [300,301])
```

Pattern

class **Pattern**(*function*, *time=1*, *arg=None*) [source]

Periodically calls a Python function.

The play() method starts the pattern timer and is not called at the object creation time.

Parent: PyoObject

Args: function: Python function

Python function to be called periodically.

time: float or PyoObject, optional

Time, in seconds, between each call. Default to 1.

arg: anything, optional

Argument sent to the function's call. If None, the function will be called without argument. Defaults to None.

Note: The out() method is bypassed. Pattern doesn't return signal.
Pattern has no *mul* and *add* attributes.

If *arg* is None, the function must be defined without argument:

```
>>> def tocall():
>>>     # function's body
```

If *arg* is not None, the function must be defined with one argument:

```
>>> def tocall(arg):
>>>     print(arg)
```

```
>>> s = Server().boot()
>>> s.start()
>>> t = HarmTable([1,0,.33,0,.2,0,.143,0,.111])
>>> a = Osc(table=t, freq=[250,251], mul=.2).out()
>>> def pat():
...     f = random.randrange(200, 401, 25)
...     a.freq = [f, f+1]
>>> p = Pattern(pat, .125)
>>> p.play()
```

setFunction(*x*) [source]

Replace the *function* attribute.

Args: x: Python function

new *function* attribute.

`setTime`(*x*) [source]

Replace the *time* attribute.

Args: x: float or PyoObject
New *time* attribute.

`setArg`(*x*) [source]

Replace the *arg* attribute.

Args: x: Anything
new *arg* attribute.

`function`

Python function. Function to be called.

`time`

float or PyoObject. Time, in seconds, between each call.

`arg`

Anything. Callable's argument.

Score

class `Score`(*input*, *fname='event_'*) [source]

Calls functions by incrementation of a preformatted name.

Score takes audio stream containning integers in input and calls a function whose name is the concatenation of *fname* and the changing integer.

Can be used to sequence events, first by creating functions p0, p1, p2, etc. and then, by passing a counter to a Score object with "p" as *fname* argument. Functions are called without parameters.

Parent: `PyoObject`

Args: input: PyoObject
Audio signal. Must contains integer numbers. Integer must change before calling its function again.

fname: string, optional
Name of the functions to be called. Defaults to "event_", meaning that the object will call the function "event_0", "event_1", "event_2", and so on... Available at initialization time only.

Note: The out() method is bypassed. Score's signal can not be sent to audio outs.
Score has no *mul* and *add* attributes.

See also: Pattern, TrigFunc

```
>>> s = Server().boot()
>>> s.start()
>>> a = SineLoop(freq=[200,300,400,500], feedback=0.05, mul=.1).out()
>>> def event_0():
...     a.freq=[200,300,400,500]
>>> def event_1():
...     a.freq=[300,400,450,600]
>>> def event_2():
...     a.freq=[150,375,450,525]
>>> m = Metro(1).play()
>>> c = Counter(m, min=0, max=3)
>>> sc = Score(c)
```

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.

input

PyoObject. Audio signal sending integer numbers.