alsaseq – ALSA sequencer bindings for Python

alsaseq is a Python module that allows to interact with ALSA sequencer clients. It can create an ALSA client, connect to other clients, send and receive ALSA events immediately or at a scheduled time using a sequencer queue. It provides a subset of the ALSA sequencer capabilities in a simplified model. It is implemented in C language and licensed under the Gnu GPL license version 2 or later. Current version is 0.1, first public release made on September 9th, 2007.

Contents

Installation

Download the **alsaseq.tar.gz** file into some directory:

```
$ wget http://pp.com.mx/python/alsaseq/alsaseq-0.1.tar.gz
```

Extract the contents:

```
$ tar xf alsaseq-0.1.tar.gz
```

Enter the folder alsaseq that was created:

```
$ cd alsaseq-0.1
```

Compile the module:

```
$ gcc -shared -I /usr/include/python2.5 -lasound -o alsaseq.so alsaseq.c.
```

You need the Python and ALSA development header files (install **python-dev** and **libasound2-dev** for Ubuntu; **libpython2.4-devel** and **libalsa2-devel** for Mandriva) and the GNU compiler.

Copy the **alsaseq.so** file to /usr/local/bin, and **alsamidi.py** to /usr/local/lib/python2.5/site-packages.

Interactive use

Create an ALSA sequencer client with one input port and one output port, no queue:

```
>>> import alsaseq
>>> alsaseq.client( 'Simple', 1, 1, False )
```

Port 0 is input, port 1 is output. Connect ALSA client 129 (could be a musical keyboard or Virtual Keyboard) to the input port, and connect output port to ALSA client 130 (a MIDI to sound converter like Timidity):

```
>>> alsaseq.connectfrom( 0, 129, 0 )
>>> alsaseq.connectto( 1, 130, 0 )
```

Check if there are events in the input port due to notes played in the MIDI keyboard:

```
>>> alsaseq.inputpending()
2
```

Read and display an ALSA event:

```
>>> alsaseq.input()
(6, 1, 0, 253, (0, 0), (0, 0), (0, 0), (0, 60, 127, 0, 0))
```

ALSA events are tuples with 8 elements: (type, flags, tag, queue, time stamp, source, destination, data). In a client without queue, received events have no time information and are asigned dummy queue number 253.

Save a received event to send it for immediate processing:

```
>>> event = alsaseq.input()
>>> alsaseq.output( event )
```

In a client with createqueue = True, you first start the queue in order to schedule events for execution at a certain time:

In order to time stamp received events and to schedule output events for execution at a certain time, specify the client with createqueue = True, and start the queue :

```
>>> alsaseq.client( 'Simple', 1, 1, True )
>>> alsaseq.start()
>>> alsaseq.output( (6, 1, 0, 1, (5, 0), (0, 0), (0, 0), (0, 60, 127, 0, 100)) )
```

The above output() schedules a C note at 5 seconds after the start() command, for a duration of 100ms.

```
>>> alsaseq.input()
(6, 1, 0, 1, ( 12, 125433), (0, 0), (0, 0), (0, 62, 120, 0, 100)) )
```

The above ALSA event was received in the input port 12 seconds and 123,433 millionths after the start() command. You can view the status (running, stopped) of que queue, and the current time, and stop it:

```
>>> alsaseq.status()
( 1, ( 20, 546234 ) )
>>> alsaseq.stop()
```

The status() shows the queue as running at 20 seconds, 546,234 millionths.

ALSA events for common MIDI events can be created using helper functions in alsamidi module:

```
>>> alsamidi.noteevent( 1, 60, 120, 5000, 10 )
(5, 1, 0, 0, (5, 0), (0, 0), (0, 0), (1, 60, 120, 0, 10))
```

See help(alsamidi) or pydoc(alsamidi) for more information.

Examples

In these example scripts it is assumed that client 129 is a MIDI keyboard, clients 130 and 131 are MIDI sound generation modules.

MIDI through

```
import alsaseq
alsaseq.client( 'MIDI through', 1, 1, False )
alsaseq.connectfrom( 1, 129, 0 )
alsaseq.connectto( 0, 130, 0 )
while 1:
  if alsaseq.inputpending():
     ev = alsaseq.input()
     alsaseq.output( ev )
```

Press control + c to interrupt the loop.

MIDI router

```
import alsaseq
alsaseq.client( 'Router', 1, 2, False )
alsaseq.connectfrom( 1, 129, 0 )
alsaseq.connectto( 0, 130, 0 )
alsaseq.connectto( 0, 131, 0 )

while 1:
  if alsaseq.inputpending():
     ev = list( alsaseq.input() )
     if ev[7][1] > 60:  # if note is above C split limit,
         ev[5][0] = 2,  # use second output port
     alsaseq.output( ev )
```

Recorder

```
import alsaseq, pickle
alsaseq.client( 'Recorder', 1, 0, True )
alsaseq.connectfrom( 1, 129, 0 )
alsaseq.start()
events = []
while 1:
    if alsaseq.inputpending():
        event = alsaseq.input()
        if event[7][1] == 56: # if note is central G#
```

```
            break             # quit recording
        events.append( ev )


pickle.dump( events, open( 'events.seq', 'w' ) )
```

Player

```
import alsaseq, pickle
events = pickle.load( open( ruta ) )

alsaseq.client( 'Player', 0, 1, True )
alsaseq.connectto( 0, 130, 0 )
alsaseq.start()

for event in events:
    alsaseq.output( event )
```