
Pyfmodex

Bart Van Loon

May 18, 2021

CONTENTS:

1	About	3
2	Installation	5
3	Quickstart	7
4	Examples	9
5	Indices and tables	67

This is pyfmodex, an [FMOD](#) Python binding using [ctypes](#). While the FMOD library is not Source, this Python binding is available under the MIT license from [GitHub](#) or [PyPI](#).

1.1 FMOD

FMOD is a solution of adaptive audio, mostly used for games. The software suite consists of two components:

FMOD Studio: A GUI to build adaptive audio.

FMOD Engine: An API to play adaptive audio.

The FMOD Engine API consists of

Studio API: Plays back content created within the FMOD Studio authoring tool. Studio's data-driven approach means audio behaviors remain easily accessible and editable to sound designers.

Core API: Allows for custom requirements that go beyond what the FMOD Studio API offers, providing fast and flexible access to low-level audio primitives.

The documentation for these components can be found at <https://fmod.com/resources/documentation-api>.

1.2 pyfmodex

The FMOD APIs have official bindings for C, C++, C# and Javascript. *pyfmodex* provides unofficial bindings for Python.

1.2.1 Goal of the project

The goal of the *pyfmodex* project - ran by volunteers in the Open Source community - is to provide a first class package to allow users to interface with FMOD from within their Python programs without needing to worry about the internals.

The intention is to

- support the last three stable minor releases of Python
- support Linux x86 and Windows platforms
- keep documentation and code quality consistently high

Given the portable nature of both Python and the FMOD libraries, this ought to mean that *pyfmodex* works just fine on other platforms (Mac OS) or architectures (Raspberry Pi, ...).

This concerted effort can be found on the [Github page](#) of the project's founder Lukáš Tyrychtr.

1.2.2 The name

The name *pyfmodex* comes from the legacy name “FMOD Ex” that was used by FMOD in the past.

INSTALLATION

To install, first make sure that you have the FMOD Engine library for your platform somewhere in your path, so Python will be able to find it. On Linux, libraries are searched for in `LD_LIBRARY_PATH`.

To download the FMOD Engine library, visit <http://www.fmod.org/download>. The library is free to download, but requires a free account to be made first.

Then, install `pyfmodex` via `pip`, `easy_install` or the `setup.py` way.

To verify if everything works, open a Python REPL and try importing `pyfmodex`:

```
import pyfmodex
```

If there is no error: good, it worked. :-)

QUICKSTART

Let's play a sample sound. Try the following simple script:

```
1 import pyfmodex
2
3 system = pyfmodex.System()
4 system.init()
5 sound = system.create_sound("somefile.mp3")
6 channel = sound.play()
7
8 while channel.is_playing:
9     pass
10
11 sound.release()
12 system.release()
```

Of course, *somefile.mp3* must be replaced with something that actually exists. :-)

Note that the while loop is necessary (at least in this simple example) to keep the main thread alive long enough. You should know this if you want to use FMOD however. If you don't, it's probably a good thing to spend some time with the [FMOD API documentation](#) first.

EXAMPLES

While we strive to keep the number of external dependencies required to run the examples below small (ideally: zero), many of the examples require the `curses` library. On Windows, this unfortunately requires the installation of an extra module, for example `windows-curses`.

4.1 Device detection

This is a sample script pretty printing the audio and recording devices detected by the FMOD Engine on your system.

```
1  """Sample code to list identification information about all sound devices
2  (audio out and audio in) detected by FMOD Engine on a system.
3  """
4
5  import re
6
7  import pyfmodex
8  from pyfmodex.enums import RESULT, SPEAKERMODE
9  from pyfmodex.exceptions import FmodError
10 from pyfmodex.flags import DRIVER_STATE
11
12 system = pyfmodex.System()
13 system.init()
14
15
16 def _pp_driverinfo(driverinfo, indent=1):
17     """Pretty print driverinfo.
18
19     Lists all keys in the given pyfmodex.structobject with their values,
20     indented by the given number times four spaces.
21
22     .. todo:: Figure out how the GUID structure works exactly.
23     """
24     for key in driverinfo.keys():
25         value = driverinfo[key]
26         if isinstance(value, bytes):
27             value = value.decode()
28         elif isinstance(value, pyfmodex.structures.GUID):
29             continue
30         elif key == "system_rate":
31             value = f"{value} kHz"
```

(continues on next page)

(continued from previous page)

```

32     elif key == "speaker_mode":
33         value = SPEAKERMODE(value).name
34     elif key == "state":
35         value = re.sub(r"^DRIVER_STATE.|\\)$", "", str(DRIVER_STATE(value))).replace(
36             "|", ", "
37         )
38     print(4 * " " * indent, end="")
39     print(f"{key}: {value}")
40 print()
41
42
43 def list_drivers(title, meth):
44     """List and prettyprint information about drivers returned by the given
45     method.
46     """
47     print(title)
48     print("-" * len(title))
49     counter = 0
50     while True:
51         try:
52             driverinfo = meth(counter)
53         except FmodError as fmoderr:
54             if fmoderr.result == RESULT.INVALID_PARAM:
55                 break
56             raise fmoderr
57         print(f"Index {counter}:")
58         _pp_driverinfo(driverinfo)
59         counter += 1
60
61
62 list_drivers("Detected audio OUT devices", system.get_driver_info)
63 list_drivers("Detected audio IN devices", system.get_record_driver_info)
64
65 system.release()

```

4.2 3D sound positioning

This is a sample script demonstrating the very basics of 3D sound positioning.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Sample code to show basic positioning of 3D sounds."""
2
3  import curses
4  import sys
5  import time
6  from math import sin
7
8  import pyfmodex
9  from pyfmodex.flags import MODE

```

(continues on next page)

(continued from previous page)

```

10
11 INTERFACE_UPDATETIME = 50
12 DISTANCEFACTOR = 1
13 MIN_FMOD_VERSION = 0x00020108
14
15 # Create system object and initialize
16 system = pyfmodex.System()
17 VERSION = system.version
18 if VERSION < MIN_FMOD_VERSION:
19     print(
20         f"FMOD lib version {VERSION:#08x} doesn't meet "
21         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
22     )
23     sys.exit(1)
24
25 system.init(maxchannels=3)
26
27 THREED_SETTINGS = system.threed_settings
28 THREED_SETTINGS.distance_factor = DISTANCEFACTOR
29
30 # Load some sounds
31 sound1 = system.create_sound("media/drumloop.wav", mode=MODE.THREED)
32 sound1.min_distance = 0.5 * DISTANCEFACTOR
33 sound1.max_distance = 5000 * DISTANCEFACTOR
34 sound1.mode = MODE.LOOP_NORMAL
35
36 sound2 = system.create_sound("media/jaguar.wav", mode=MODE.THREED)
37 sound2.min_distance = 0.5 * DISTANCEFACTOR
38 sound2.max_distance = 5000 * DISTANCEFACTOR
39 sound2.mode = MODE.LOOP_NORMAL
40
41 sound3 = system.create_sound("media/swish.wav")
42
43 # Play sounds at certain positions
44 channel1 = system.play_sound(sound1, paused=True)
45 channel1.position = (-10 * DISTANCEFACTOR, 0, 0)
46 channel1.paused = False
47
48 channel2 = system.play_sound(sound2, paused=True)
49 channel2.position = (15 * DISTANCEFACTOR, 0, 0)
50 channel2.paused = False
51
52 # Main loop
53 def main(stdscr):
54     """Draw a simple TUI, grab keypresses and let the user manipulate a simple
55     environment with a listener and some sounds.
56     """
57     listener = system.listener(0)
58     pos_ch1 = int((channel1.position[0]) / DISTANCEFACTOR) + 25
59     pos_ch2 = int((channel2.position[0]) / DISTANCEFACTOR) + 25
60
61     stdscr.clear()

```

(continues on next page)

```

62 stdscr.nodelay(True)
63
64 # Create small visual display
65 stdscr.addstr(
66     "=====\n"
67     "3D Example.\n"
68     "=====\n"
69     "\n"
70     "Press 1 to toggle sound 1 (16bit Mono 3D)\n"
71     "Press 2 to toggle sound 2 (8bit Mono 3D)\n"
72     "Press 3 to play a sound (16bit Stereo 2D)\n"
73     "Press h or l to move listener (when in still mode)\n"
74     "Press space to toggle listener still mode\n"
75     "Press q to quit"
76 )
77
78 listener_automove = True
79 listener_prevposx = 0
80 listener_velx = 0
81 clock = 0
82 while True:
83     tic = time.time()
84
85     listener_posx = listener.position[0]
86     environment = list("|" + 48 * "." + "|")
87     environment[pos_ch1 - 1 : pos_ch1 + 2] = list("<1>")
88     environment[pos_ch2 - 1 : pos_ch2 + 2] = list("<2>")
89     environment[int(listener_posx / DISTANCEFACTOR) + 25] = "L"
90
91     stdscr.addstr(11, 0, "".join(environment))
92     stdscr.addstr("\n")
93
94     # Listen to the user
95     try:
96         keypress = stdscr.getkey()
97         if keypress == "1":
98             channel1.paused = not channel1.paused
99         elif keypress == "2":
100             channel2.paused = not channel2.paused
101         elif keypress == "3":
102             system.play_sound(sound3)
103         elif keypress == " ":
104             listener_automove = not listener_automove
105         elif keypress == "q":
106             break
107
108         if not listener_automove:
109             if keypress == "h":
110                 listener_posx = max(
111                     -24 * DISTANCEFACTOR, listener_posx - DISTANCEFACTOR
112                 )
113             elif keypress == "l":

```

(continues on next page)

(continued from previous page)

```

114         listener_posx = min(
115             23 * DISTANCEFACTOR, listener_posx + DISTANCEFACTOR
116         )
117     except curses.error as cerr:
118         if cerr.args[0] != "no input":
119             raise cerr
120
121     # Update the listener
122     if listener_automove:
123         listener_posx = sin(clock * 0.05) * 24 * DISTANCEFACTOR
124         listener_velx = (listener_posx - listener_prevposx) * (
125             1000 / INTERFACE_UPDATETIME
126         )
127
128         listener.position = (listener_posx, 0, 0)
129         listener.velocity = (listener_velx, 0, 0)
130         listener_prevposx = listener_posx
131
132         clock += 30 * (1 / INTERFACE_UPDATETIME)
133         system.update()
134
135         toc = time.time()
136         time.sleep(max(0, INTERFACE_UPDATETIME / 1000 - (toc - tic)))
137
138
139 curses.wrapper(main)
140
141 # Shut down
142 sound1.release()
143 sound2.release()
144 sound3.release()
145
146 system.release()

```

4.3 Channel groups

This is sample script showing how to put channels into channel groups.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Sample code to show how to put channels into channel groups."""
2
3  import curses
4  import sys
5  import time
6
7  import pyfmodex
8  from pyfmodex.flags import MODE
9
10 MIN_FMOD_VERSION = 0x00020108

```

(continues on next page)

```
11
12 # Create a System object and initialize
13 system = pyfmodex.System()
14 VERSION = system.version
15 if VERSION < MIN_FMOD_VERSION:
16     print(
17         f"FMOD lib version {VERSION:#08x} doesn't meet "
18         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
19     )
20     sys.exit(1)
21
22 system.init(maxchannels=6)
23
24
25 # Load some sounds
26 sounds = []
27 sounds.append(system.create_sound("media/drumloop.wav", mode=MODE.LOOP_NORMAL))
28 sounds.append(system.create_sound("media/jaguar.wav", mode=MODE.LOOP_NORMAL))
29 sounds.append(system.create_sound("media/swish.wav", mode=MODE.LOOP_NORMAL))
30 sounds.append(system.create_sound("media/c.ogg", mode=MODE.LOOP_NORMAL))
31 sounds.append(system.create_sound("media/d.ogg", mode=MODE.LOOP_NORMAL))
32 sounds.append(system.create_sound("media/e.ogg", mode=MODE.LOOP_NORMAL))
33
34 group_a = system.create_channel_group("Group A")
35 group_b = system.create_channel_group("Group B")
36 group_master = system.master_channel_group
37
38 # Instead of being independent, set the group A and B to be children of the
39 # master group
40 group_master.add_group(group_a)
41 group_master.add_group(group_b)
42
43 # Start all the sounds
44 for idx, sound in enumerate(sounds):
45     system.play_sound(sound, channel_group=group_a if idx < 3 else group_b)
46
47 # Change the volume of each group, just because we can! (reduce overall noise)
48 group_a.volume = 0.5
49 group_b.volume = 0.5
50
51 # Main loop
52 def main(stdscr):
53     """Draw a simple TUI, grab keypresses and let the user manipulate the
54     channel groups.
55     """
56     stdscr.clear()
57     stdscr.nodelay(True)
58
59     # Create small visual display
60     stdscr.addstr(
61         "=====\n"
62         "Channel Groups Example.\n"
```

(continues on next page)

(continued from previous page)

```

63     "=====\n"
64     "\n"
65     "Group A : drumloop.wav, jaguar.wav, swish.wav\n"
66     "Group B : c.ogg, d.ogg, e.ogg\n"
67     "\n"
68     "Press a to mute/unmute group A\n"
69     "Press b to mute/unmute group B\n"
70     "Press m to mute/unmute master group\n"
71     "Press q to quit"
72 )
73
74 while True:
75     stdscr.addstr(
76         12, 0, f"Channels playing: {system.channels_playing['channels']}\n"
77     )
78
79     # Listen to the user
80     try:
81         keypress = stdscr.getkey()
82         if keypress == "a":
83             group_a.mute = not group_a.mute
84         elif keypress == "b":
85             group_b.mute = not group_b.mute
86         elif keypress == "m":
87             group_master.mute = not group_master.mute
88         elif keypress == "q":
89             break
90     except curses.error as cerr:
91         if cerr.args[0] != "no input":
92             raise cerr
93
94     system.update()
95     time.sleep(50 / 1000)
96
97     # A little fade out
98     if not (group_master.mute or group_a.mute and group_b.mute):
99         pitch = 1.0
100        volume = 1.0
101
102        fadeout_sec = 3
103        for _ in range(10 * fadeout_sec):
104            group_master.pitch = pitch
105            group_master.volume = volume
106
107            volume -= 1 / (10 * fadeout_sec)
108            pitch -= 0.25 / (10 * fadeout_sec)
109
110            system.update()
111            time.sleep(0.1)
112
113
114 curses.wrapper(main)

```

(continues on next page)

(continued from previous page)

```

115
116 # Shut down
117 for sound in sounds:
118     sound.release()
119
120 group_a.release()
121 group_b.release()
122
123 system.release()

```

4.4 Convolution reverb

This is a sample script showing how to set up a convolution reverb DSP and work with it.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Sample code to demonstrate how to set up a convolution reverb DSP and work
2  with it.
3  """
4
5  import curses
6  import sys
7  import time
8  from ctypes import c_short, sizeof
9
10 import pyfmodex
11 from pyfmodex.enums import (CHANNELCONTROL_DSP_INDEX, DSP_CONVOLUTION_REVERB,
12                             DSP_TYPE, DSPCONNECTION_TYPE, SOUND_FORMAT,
13                             TIMEUNIT)
14 from pyfmodex.flags import MODE
15
16 MIN_FMOD_VERSION = 0x00020108
17
18 # Create a System object and initialize
19 system = pyfmodex.System()
20 VERSION = system.version
21 if VERSION < MIN_FMOD_VERSION:
22     print(
23         f"FMOD lib version {VERSION:#08x} doesn't meet "
24         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
25     )
26     sys.exit(1)
27
28 system.init(maxchannels=1)
29
30 # Create a new channel group to hold the convolution DSP unit
31 reverbgroup = system.create_channel_group("reverb")
32
33 # Create a new channel group to hold all the channels and process the dry path
34 maingroup = system.create_channel_group("main")

```

(continues on next page)

(continued from previous page)

```

35
36 # Create the convolution DSP unit and set it as the tail of the channel group
37 reverbunit = system.create_dsp_by_type(DSP_TYPE.CONVOLUTIONREVERB)
38 reverbgroup.add_dsp(CHANNELCONTROL_DSP_INDEX.TAIL, reverbunit)
39
40 # Open the impulse response wav file, but use FMOD_OPENONLY as we want to read
41 # the data into a separate buffer
42 irsound = system.create_sound("media/standrews.wav", mode=MODE.DEFAULT | MODE.OPENONLY)
43
44 # For simplicity of the example, if the impulse response is the wrong format
45 # just display an error
46 if irsound.format.format != SOUND_FORMAT.PCM16:
47     print(
48         "Impulse Response file is the wrong audio format. It should be 16bit"
49         " integer PCM data."
50     )
51     sys.exit(1)
52
53 # The reverb unit expects a block of data containing a single 16 bit int
54 # containing the number of channels in the impulse response, followed by PCM 16
55 # data
56 short_size = sizeof(c_short)
57 irsound_channels = irsound.format.channels
58 irsound_data_length = irsound.get_length(TIMEUNIT.PCMBYTES)
59 irdata = (c_short * (1 + irsound_data_length))()
60 irsound_data = irsound.read_data(irsound_data_length)[0]
61
62 irdata[0] = irsound_channels
63 irdata[1:] = list(irsound_data)
64
65 reverbunit.set_parameter_data(DSP_CONVOLUTION_REVERB.PARAM_IR, irdata)
66
67 # Don't pass any dry signal from the reverb unit, instead take the dry part of
68 # the mix from the main signal path
69 reverbunit.set_parameter_float(DSP_CONVOLUTION_REVERB.PARAM_DRY, -80)
70
71 # We can now release the sound object as the reverb unit has created its
72 # internal data
73 irsound.release()
74
75 # Load up and play a sample clip recorded in an anechoic chamber
76 sound = system.create_sound("media/singing.wav", mode=MODE.THREED | MODE.LOOP_NORMAL)
77 channel = system.play_sound(sound, channel_group=maingroup, paused=True)
78
79 # Create a send connection between the channel head and the reverb unit
80 channel_head = channel.get_dsp(CHANNELCONTROL_DSP_INDEX.HEAD)
81 reverb_connection = reverbunit.add_input(channel_head, DSPCONNECTION_TYPE.SEND)
82
83 channel.paused = False
84
85 # Main loop
86 def main(stdscr):

```

(continues on next page)

```

87  """Draw a simple TUI, grab keypresses and let the user manipulate the
88  reverb connection.
89  """
90  wet_volume = 1
91  dry_volume = 1
92
93  stdscr.clear()
94  stdscr.nodelay(True)
95
96  # Create small visual display
97  stdscr.addstr(
98      "=====\n"
99      "Convolution Example.\n"
100     "=====\n"
101     "\n"
102     "Press k and j to change dry mix\n"
103     "Press h and l to change wet mix\n"
104     "Press q to quit"
105 )
106
107 while True:
108     stdscr.addstr(8, 0, f"wet mix [{wet_volume:.2f}] | dry mix [{dry_volume:.2f}]")
109
110     # Listen to the user
111     try:
112         keypress = stdscr.getkey()
113         if keypress == "h":
114             wet_volume = max(wet_volume - 0.05, 0)
115         elif keypress == "l":
116             wet_volume = min(wet_volume + 0.05, 1)
117         elif keypress == "j":
118             dry_volume = max(dry_volume - 0.05, 0)
119         elif keypress == "k":
120             dry_volume = min(dry_volume + 0.05, 1)
121         elif keypress == "q":
122             break
123     except curses.error as cerr:
124         if cerr.args[0] != "no input":
125             raise cerr
126
127     reverb_connection.mix = wet_volume
128     maingroup.volume = dry_volume
129
130     system.update()
131     time.sleep(50 / 1000)
132
133 curses.wrapper(main)
134
135 # Shut down
136 sound.release()
137 maingroup.release()
138

```

(continues on next page)

(continued from previous page)

```

139 reverbgroup.remove_dsp(reverbunit)
140 reverbunit.disconnect_all(inputs=True, outputs=True)
141 reverbunit.release()
142 reverbgroup.release()
143 system.release()

```

4.5 DSP effect per speaker

This is a sample script showing how to manipulate a DSP network and as an example, creating two DSP effects, splitting a single sound into two audio paths, which then gets filtered separately.

To only have each audio path come out of one speaker each, `set_mix_matrix()` is used just before the two branches merge back together again.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to demonstrate how to manipulate DSP network to have two
2  different effects on seperately filtered, different audio paths from a single
3  sound.
4  """
5
6  import curses
7  import sys
8  import time
9
10 import pyfmodex
11 from pyfmodex.enums import (CHANNELCONTROL_DSP_INDEX, DSP_MULTIBAND_EQ,
12                             DSP_MULTIBAND_EQ_FILTER_TYPE, DSP_TYPE,
13                             SPEAKERMODE)
14 from pyfmodex.flags import MODE
15 from pyfmodex.structobject import Structobject
16
17 MIN_FMOD_VERSION = 0x00020108
18
19 # Create a System object and initialize
20 system = pyfmodex.System()
21 VERSION = system.version
22 if VERSION < MIN_FMOD_VERSION:
23     print(
24         f"FMOD lib version {VERSION:#08x} doesn't meet "
25         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
26     )
27     sys.exit(1)
28
29 # In this special case we want to use stereo output and not worry about varying
30 # matrix sizes depending on user speaker mode
31 software_format = Structobject(
32     sample_rate=48000, speaker_mode=SPEAKERMODE.STEREO, raw_speakers=0
33 )
34 system.software_format = software_format
35

```

(continues on next page)

```

36 # Initialize FMOD
37 system.init(maxchannels=1)
38
39 sound = system.create_sound("media/drumloop.wav", mode=MODE.LOOP_NORMAL)
40 channel = system.play_sound(sound)
41
42 # Create the DSP effects
43 dsplowpass = system.create_dsp_by_type(DSP_TYPE.MULTIBAND_EQ)
44 dsplowpass.set_parameter_int(
45     DSP_MULTIBAND_EQ.A_FILTER, DSP_MULTIBAND_EQ_FILTER_TYPE.LOWPASS_24DB
46 )
47 dsplowpass.set_parameter_float(DSP_MULTIBAND_EQ.A_FREQUENCY, 1000)
48 dsplowpass.set_parameter_float(DSP_MULTIBAND_EQ.A_Q, 4)
49
50 dsphighpass = system.create_dsp_by_type(DSP_TYPE.MULTIBAND_EQ)
51 dsphighpass.set_parameter_int(
52     DSP_MULTIBAND_EQ.A_FILTER, DSP_MULTIBAND_EQ_FILTER_TYPE.HIGHPASS_24DB
53 )
54 dsphighpass.set_parameter_float(DSP_MULTIBAND_EQ.A_FREQUENCY, 4000)
55 dsphighpass.set_parameter_float(DSP_MULTIBAND_EQ.A_Q, 4)
56
57 # Connect up the DSP network
58
59 # When a sound is played, a subnetwork is set up in the DSP network which looks
60 # like this (wavetable is the drumloop sound, and it feeds its data from right
61 # to left):
62 #
63 # [DSPHEAD]<---[DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
64 group_master = system.master_channel_group
65 dsphead = group_master.get_dsp(CHANNELCONTROL_DSP_INDEX.HEAD)
66 dspchannelmixer, _ = dsphead.get_input(0)
67
68 # Now disconnect channeldsp head from the wavetable to make it look like this:
69 #
70 # [DSPHEAD] [DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
71 dsphead.disconnect_from(dspchannelmixer)
72
73 # Now connect the two effects to channeldsp head and store the two connections
74 # this makes so we can set their matrix later
75
76 # [DSPLOWPASS]
77 # /x
78 # [DSPHEAD] [DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
79 # \y
80 # [DSPHIGHPASS]
81 dsplowpassconnection = dsphead.add_input(dsplowpass) # x
82 dsphighpassconnection = dsphead.add_input(dsphighpass) # y
83
84 # Now connect the channelmixer to the 2 effects
85 # [DSPLOWPASS]
86 # /x \
87 # [DSPHEAD] [DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]

```

(continues on next page)

(continued from previous page)

```

88 #           \y           /
89 #           [DSPHIGHPASS]
90
91 dsplowpass.add_input(dspchannelmixer) # Ignore connection - we dont care about it.
92 dsphighpass.add_input(dspchannelmixer) # Ignore connection - we dont care about it.
93
94 # Now the drumloop will be twice as loud, because it is being split into 2,
95 # then recombined at the end. What we really want is to only feed the
96 # dsphead<-dsplowpass through the left speaker for that effect, and
97 # dsphead<-dsphighpass to the right speaker for that effect. We can do that
98 # simply by setting the pan, or speaker matrix of the connections
99
100 #           [DSPLOWPASS]
101 #           /x=1,0           \
102 # [DSPHEAD]           [DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
103 #           \y=0,1           /
104 #           [DSPHIGHPASS]
105
106 lowpassmatrix = [
107     1, 0, # output to front left: take front left input signal at 1
108     0, 0, # output to front right: silence
109 ]
110 highpassmatrix = [
111     0, 0, # output to front left: silence
112     0, 1, # output to front right: take front right input signal at 1
113 ]
114
115 # Upgrade the signal coming from the channel mixer from mono to stereo
116 # Otherwise the lowpass and highpass will get mono signals
117 dspchannelmixer.channel_format = Structobject(
118     channel_mask=0, num_channels=0, source_speaker_mode=SPEAKERMODE.STEREO
119 )
120
121 # Now set the above matrices
122 dsplowpassconnection.set_mix_matrix(lowpassmatrix, 2, 2)
123 dsphighpassconnection.set_mix_matrix(highpassmatrix, 2, 2)
124
125 dsplowpass.bypass = True
126 dsphighpass.bypass = True
127
128 dsplowpass.active = True
129 dsphighpass.active = True
130
131 # Main loop
132 def main(stdscr):
133     """Draw a simple TUI, grab keypresses and let the user manipulate the
134     DSP states.
135     """
136     pan = 0
137
138     stdscr.clear()
139     stdscr.nodelay(True)

```

(continues on next page)

```

140
141 # Create small visual display
142 stdscr.addstr(
143     "=====\n"
144     "DSP Effect Per Speaker Sample.\n"
145     "=====\n"
146     "\n"
147     "Press 1 to toggle lowpass (left speaker)\n"
148     "Press 2 to toggle highpass (right speaker)\n"
149     "Press h and l to pan sound\n"
150     "Press q to quit"
151 )
152
153 while True:
154     stdscr.addstr(
155         10,
156         0,
157         f"Lowpass (left) is {'inactive' if dsplowpass.bypass else 'active '}",
158     )
159     stdscr.addstr(
160         11,
161         0,
162         f"Highpass (right) is {'inactive' if dsphighpass.bypass else 'active '}",
163     )
164     stdscr.addstr(12, 0, f"Pan is {pan:.1f} ")
165
166 # Listen to the user
167 try:
168     keypress = stdscr.getkey()
169     if keypress == "1":
170         dsplowpass.bypass = not dsplowpass.bypass
171     elif keypress == "2":
172         dsphighpass.bypass = not dsphighpass.bypass
173     elif keypress == "h":
174         pan = max(pan - 0.1, -1)
175         channel.set_pan(pan)
176     elif keypress == "l":
177         pan = min(pan + 0.1, 1)
178         channel.set_pan(pan)
179     elif keypress == "q":
180         break
181 except curses.error as cerr:
182     if cerr.args[0] != "no input":
183         raise cerr
184
185     system.update()
186     time.sleep(50 / 1000)
187
188 curses.wrapper(main)
189
190 # Shut down
191

```

(continues on next page)

(continued from previous page)

```

192 sound.release()
193 dsplowpass.release()
194 dsphighpass.release()
195 system.release()

```

4.6 Effects

This is a sample script showing how to apply some of the built in software effects to sounds by applying them to the master channel group. All software sounds played here would be filtered in the same way. To filter per channel, and not have other channels affected, simply apply the same function to the `Channel` instead of the `ChannelGroup`.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to apply some built in software effects to sounds.
2  """
3
4  import curses
5  import sys
6  import time
7
8  import pyfmodex
9  from pyfmodex.enums import (DSP_MULTIBAND_EQ, DSP_MULTIBAND_EQ_FILTER_TYPE,
10                             DSP_TYPE)
11
12  MIN_FMOD_VERSION = 0x00020108
13
14  # Create a System object and initialize.
15  system = pyfmodex.System()
16  VERSION = system.version
17  if VERSION < MIN_FMOD_VERSION:
18      print(
19          f"FMOD lib version {VERSION:#08x} doesn't meet "
20          f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
21      )
22      sys.exit(1)
23
24  system.init(maxchannels=1)
25
26  mastergroup = system.master_channel_group
27  sound = system.create_sound("media/drumloop.wav")
28  channel = system.play_sound(sound)
29
30  # Create some effects to play with
31  dsplowpass = system.create_dsp_by_type(DSP_TYPE.MULTIBAND_EQ)
32  dsphighpass = system.create_dsp_by_type(DSP_TYPE.MULTIBAND_EQ)
33  dspecho = system.create_dsp_by_type(DSP_TYPE.ECHO)
34  dspflange = system.create_dsp_by_type(DSP_TYPE.FLANGE)
35
36  # Configure multiband_eq DSPs to create lowpass and highpass filters
37  dsplowpass.set_parameter_int(

```

(continues on next page)

(continued from previous page)

```

38     DSP_MULTIBAND_EQ.A_FILTER, DSP_MULTIBAND_EQ_FILTER_TYPE.LOWPASS_24DB
39 )
40 dsplowpass.set_parameter_float(DSP_MULTIBAND_EQ.A_FREQUENCY, 1000)
41 dsplowpass.set_parameter_float(DSP_MULTIBAND_EQ.A_Q, 4)
42
43 dsphighpass.set_parameter_int(
44     DSP_MULTIBAND_EQ.A_FILTER, DSP_MULTIBAND_EQ_FILTER_TYPE.HIGHPASS_24DB
45 )
46 dsphighpass.set_parameter_float(DSP_MULTIBAND_EQ.A_FREQUENCY, 4000)
47 dsphighpass.set_parameter_float(DSP_MULTIBAND_EQ.A_Q, 4)
48
49 # Add them to the master channel group. Each time an effect is added (to
50 # position 0) it pushes the others down the list.
51 mastergroup.add_dsp(0, dsplowpass)
52 mastergroup.add_dsp(0, dsphighpass)
53 mastergroup.add_dsp(0, dspecho)
54 mastergroup.add_dsp(0, dspflange)
55
56 # By default, bypass all effects. This means let the original signal go
57 # through without processing. It will sound 'dry' until effects are enabled by
58 # the user.
59 dsplowpass.bypass = True
60 dsphighpass.bypass = True
61 dspecho.bypass = True
62 dspflange.bypass = True
63
64 # Main loop
65 def main(stdscr):
66     """Draw a simple TUI, grab keypresses and let the user manipulate the
67     DSP states.
68     """
69     stdscr.clear()
70     stdscr.nodelay(True)
71
72     # Create small visual display
73     stdscr.addstr(
74         "=====\n"
75         "Effects Example.\n"
76         "=====\n"
77         "\n"
78         "Press SPACE to pause/unpause sound\n"
79         "Press 1 to toggle dsplowpass effect\n"
80         "Press 2 to toggle dsphighpass effect\n"
81         "Press 3 to toggle dspecho effect\n"
82         "Press 4 to toggle dspflange effect\n"
83         "Press q to quit"
84     )
85
86     while True:
87         stdscr.addstr(
88             11,
89             0,

```

(continues on next page)

(continued from previous page)

```

90     "%-8s: lowpass[%s] highpass[%s] echo [%s] flange[%s]"
91     % (
92         "Paused" if channel.paused else "Playing",
93         " " if dsplowpass.bypass else "x",
94         " " if dsphighpass.bypass else "x",
95         " " if dspecho.bypass else "x",
96         " " if dspflange.bypass else "x",
97     ),
98 )
99
100 # Listen to the user
101 try:
102     keypress = stdscr.getkey()
103     if keypress == " ":
104         channel.paused = not channel.paused
105     elif keypress == "1":
106         dsplowpass.bypass = not dsplowpass.bypass
107     elif keypress == "2":
108         dsphighpass.bypass = not dsphighpass.bypass
109     elif keypress == "3":
110         dspecho.bypass = not dspecho.bypass
111     elif keypress == "4":
112         dspflange.bypass = not dspflange.bypass
113     elif keypress == "q":
114         break
115 except curses.error as cerr:
116     if cerr.args[0] != "no input":
117         raise cerr
118
119     system.update()
120     time.sleep(50 / 1000)
121
122
123 curses.wrapper(main)
124
125 # Shut down
126 mastergroup.remove_dsp(dsplowpass)
127 mastergroup.remove_dsp(dsphighpass)
128 mastergroup.remove_dsp(dspecho)
129 mastergroup.remove_dsp(dspflange)
130
131 dsplowpass.release()
132 dsphighpass.release()
133 dspecho.release()
134 dspflange.release()
135
136 sound.release()
137 system.release()

```

4.7 Gapless playback

This is a sample script showing how to schedule channel playback into the future with sample accuracy. It uses several scheduled channels to synchronize two or more sounds.

(Adapted from sample code shipped with FMOD Engine.)

```
1  """Example code to show how to schedule channel playback into the future with
2  sample accuracy. Uses several scheduled channels to synchronize two or more
3  sounds.
4  """
5
6  import curses
7  import sys
8  import time
9  from enum import IntEnum
10
11 import pyfmodex
12 from pyfmodex.enums import TIMEUNIT
13 from pyfmodex.structobject import Structobject
14
15 MIN_FMOD_VERSION = 0x00020108
16
17
18 # pylint: disable=too-few-public-methods
19 class Note(IntEnum):
20     """The notes we need to play our song."""
21
22     C = 0
23     D = 1
24     E = 2
25
26
27 SONG = [
28     Note.E, # Ma-
29     Note.D, # ry
30     Note.C, # had
31     Note.D, # a
32     Note.E, # lit-
33     Note.E, # tle
34     Note.E, # lamb,
35     Note.E, # .....
36     Note.D, # lit-
37     Note.D, # tle
38     Note.D, # lamb,
39     Note.D, # .....
40     Note.E, # lit-
41     Note.E, # tle
42     Note.E, # lamb,
43     Note.E, # .....
44     Note.E, # Ma-
45     Note.D, # ry
46     Note.C, # had
```

(continues on next page)

(continued from previous page)

```

47 Note.D, # a
48 Note.E, # lit-
49 Note.E, # tle
50 Note.E, # lamb,
51 Note.E, # its
52 Note.D, # fleece
53 Note.D, # was
54 Note.E, # white
55 Note.D, # as
56 Note.C, # snow.
57 Note.C, # .....
58 Note.C, # .....
59 Note.C, # .....
60 ]
61
62 # Create a System object and initialize.
63 system = pyfmodex.System()
64 VERSION = system.version
65 if VERSION < MIN_FMOD_VERSION:
66     print(
67         f"FMOD lib version {VERSION:#08x} doesn't meet "
68         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
69     )
70     sys.exit(1)
71
72 system.init(maxchannels=len(SONG))
73
74 # Get information needed later for scheduling: the mixer block size, and the
75 # output rate of the mixer
76
77 dsp_block_len = system.dsp_buffer_size.size
78 output_rate = system.software_format.sample_rate
79
80 # Load our sounds - these are just sine wave tones at different frequencies.
81 sounds = [None] * len(Note)
82 sounds[Note.C] = system.create_sound("media/c.ogg")
83 sounds[Note.D] = system.create_sound("media/d.ogg")
84 sounds[Note.E] = system.create_sound("media/e.ogg")
85
86 # Create a channelgroup that the channels will play on. We can use this
87 # channelgroup as our clock reference. It also means we can pause and pitch
88 # bend the channelgroup, without affecting the offsets of the delays, because
89 # the channelgroup clock which the channels feed off, will be pausing and
90 # speeding up/slowing down and still keeping the children in sync.
91 channelgroup = system.create_channel_group("Parent")
92
93 # Play all the sounds at once! Space them apart with set delay though so that
94 # they sound like they play in order.
95 CLOCK_START = 0
96 for note in SONG:
97
98     # Pick a note from our tune

```

(continues on next page)

```

99     sound = sounds[note]
100
101     # Play the sound on the channelgroup we want to use as the parent clock
102     # reference (for `delay` further down)
103     channel = system.play_sound(sound, channelgroup, paused=True)
104
105     if not CLOCK_START:
106         CLOCK_START = channel.dsp_clock.parent_clock
107
108         # Start the sound into the future, by two mixer blocks worth. Should be
109         # enough to avoid the mixer catching up and hitting the clock value
110         # before we've finished setting up everything. Alternatively the
111         # channelgroup we're basing the clock on could be paused to stop it
112         # ticking.
113         CLOCK_START += dsp_block_len * 2
114     else:
115         # Get the length of the sound in samples
116         sound_len = sound.get_length(TIMEUNIT.PCM)
117
118         # Get the default frequency that the sound was recorded at
119         freq = sound.default_frequency
120
121         # Convert the length of the sound to 'output samples' for the output
122         # timeline
123         sound_len = int(sound_len / freq * output_rate)
124
125         # Place the sound clock start time to this value after the last one
126         CLOCK_START += sound_len
127
128         # Schedule the channel to start in the future at the newly calculated
129         # channelgroup clock value
130         channel.delay = Structobject(dsp_start=CLOCK_START, dsp_end=0, stop_channels=False)
131
132         # Unpause the sound. Note that you won't hear the sounds, they are
133         # scheduled into the future.
134         channel.paused = False
135
136     # Main loop
137     def main(stdscr):
138         """Draw a simple TUI, grab keypresses and let the user manipulate the
139         channel parameters.
140         """
141         stdscr.clear()
142         stdscr.nodelay(True)
143
144         # Create small visual display
145         stdscr.addstr(
146             "=====\n"
147             "Gapless Playback example.\n"
148             "=====\n"
149             "\n"
150             "Press SPACE to toggle pause\n"

```

(continues on next page)

(continued from previous page)

```

151     "Press k to increase pitch\n"
152     "Press j to decrease pitch\n"
153     "Press q to quit"
154 )
155
156 while True:
157     paused_state = "Paused" if channelgroup.paused else "Playing"
158
159     stdscr.move(9, 0)
160     stdscr.clrtoeol()
161     stdscr.addstr(
162         f"Channels Playing {system.channels_playing.channels} : {paused_state}"
163     )
164
165     # Listen to the user
166     try:
167         keypress = stdscr.getkey()
168         if keypress == " ":
169             # Pausing the channelgroup, as the clock parent will pause any
170             # scheduled sounds from continuing. If you paused the channel,
171             # this would not stop the clock it is delayed against from
172             # ticking, and you'd have to recalculate the delay for the
173             # channel into the future again before it was unpaused.
174             channelgroup.paused = not channelgroup.paused
175         elif keypress == "k":
176             for _ in range(50):
177                 channelgroup.pitch += 0.01
178                 system.update()
179                 time.sleep(10 / 1000)
180         elif keypress == "j":
181             for _ in range(50):
182                 if channelgroup.pitch > 0.1:
183                     channelgroup.pitch -= 0.01
184                     system.update()
185                     time.sleep(10 / 1000)
186         elif keypress == "q":
187             break
188     except curses.error as cerr:
189         if cerr.args[0] != "no input":
190             raise cerr
191
192     system.update()
193     time.sleep(50 / 1000)
194
195 curses.wrapper(main)
196
197 # Shut down
198 for sound in sounds:
199     sound.release()
200 system.release()
201

```

4.8 Generate tone

This is a sample script showing how to play generated tones using `play_dsp()` instead of manually connecting and disconnecting DSP units.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to play generated tones using System.play_dsp
2  instead of manually connecting and disconnecting DSP units.
3  """
4
5  import curses
6  import sys
7  import time
8
9  import pyfmodex
10 from pyfmodex.enums import DSP_OSCILLATOR, DSP_TYPE
11
12 MIN_FMOD_VERSION = 0x00020108
13
14 # Create a System object and initialize
15 system = pyfmodex.System()
16 VERSION = system.version
17 if VERSION < MIN_FMOD_VERSION:
18     print(
19         f"FMOD lib version {VERSION:#08x} doesn't meet "
20         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
21     )
22     sys.exit(1)
23
24 system.init(maxchannels=1)
25
26 # Create an oscillator DSP unit for the tone
27 dsp = system.create_dsp_by_type(DSP_TYPE.OSCILLATOR)
28 dsp.set_parameter_float(DSP_OSCILLATOR.RATE, 440) # Musical note 'A'
29
30 # Main loop
31 def main(stdscr):
32     """Draw a simple TUI, grab keypresses and let the user manipulate the
33     DSP parameters.
34     """
35     stdscr.clear()
36     stdscr.nodelay(True)
37
38     # Create small visual display
39     stdscr.addstr(
40         "=====\n"
41         "Generate Tone Example.\n"
42         "=====\n"
43         "\n"
44         "Press 1 to play a sine wave\n"
45         "Press 2 to play a square wave\n"
46         "Press 3 to play a saw wave\n"

```

(continues on next page)

(continued from previous page)

```

47     "Press 4 to play a triangle wave\n"
48     "Press SPACE to stop the channel\n"
49     "Press q to quit\n"
50     "Press k and j to change volume\n"
51     "Press h and l to change frequency"
52 )
53
54 channel = None
55 while True:
56     if channel:
57         playing = "playing" if channel.is_playing else "stopped"
58         volume = channel.volume
59         frequency = channel.frequency
60     else:
61         playing = "stopped"
62         volume = 0
63         frequency = 0
64
65     stdscr.move(13, 0)
66     stdscr.clrtoeol()
67     stdscr.addstr(f"Channel is {playing}")
68
69     stdscr.move(14, 0)
70     stdscr.clrtoeol()
71     stdscr.addstr(f"Volume {volume:.2f}")
72
73     stdscr.move(15, 0)
74     stdscr.clrtoeol()
75     stdscr.addstr(f"Frequency {frequency}")
76
77     # Listen to the user
78     try:
79         keypress = stdscr.getkey()
80         if keypress == "1":
81             if channel:
82                 channel.stop()
83                 channel = system.play_dsp(dsp, paused=True)
84                 channel.volume = 0.5
85                 dsp.set_parameter_int(DSP_OSCILLATOR.TYPE, 0)
86                 channel.paused = False
87         elif keypress == "2":
88             if channel:
89                 channel.stop()
90                 channel = system.play_dsp(dsp, paused=True)
91                 channel.volume = 0.125
92                 dsp.set_parameter_int(DSP_OSCILLATOR.TYPE, 1)
93                 channel.paused = False
94         elif keypress == "3":
95             if channel:
96                 channel.stop()
97                 channel = system.play_dsp(dsp, paused=True)
98                 channel.volume = 0.125

```

(continues on next page)

```
99         dsp.set_parameter_int(DSP_OSCILLATOR.TYPE, 2)
100         channel.paused = False
101     elif keypress == "4":
102         if channel:
103             channel.stop()
104             channel = system.play_dsp(dsp, paused=True)
105             channel.volume = 0.5
106             dsp.set_parameter_int(DSP_OSCILLATOR.TYPE, 4)
107             channel.paused = False
108     elif keypress == " ":
109         if channel:
110             channel.stop()
111             channel = None
112     elif keypress == "q":
113         break
114
115     if not channel:
116         raise curses.error("no input")
117
118     if keypress == "h":
119         channel.frequency = max(channel.frequency - 500, 0)
120     elif keypress == "j":
121         channel.volume = max(channel.volume - 0.1, 0)
122     elif keypress == "k":
123         channel.volume = min(channel.volume + 0.1, 1)
124     elif keypress == "l":
125         channel.frequency = channel.frequency + 500
126 except curses.error as cerr:
127     if cerr.args[0] != "no input":
128         raise cerr
129
130     system.update()
131     time.sleep(50 / 1000)
132
133 curses.wrapper(main)
134
135 # Shut down
136 dsp.release()
137 system.release()
138
```

4.9 Granular synthesis

This is a sample script showing how to play a string of sounds together without gaps, using `delay` to produce a granular synthesis style truck engine effect.

The basic operation is:

1. Play two sounds initially at the same time, the first sound immediately, and the second sound with a delay calculated by the length of the first sound.
2. Set `delay` to initiate the delayed playback. The `delay` is sample accurate and uses output samples as the time

frame, not source samples. These samples are a fixed amount per second regardless of the source sound format, for example 48000 samples per second if FMOD is initialized to 48khz output.

3. Output samples are calculated from source samples with a simple source-to-output sample rate conversion.
4. When the first sound finishes, the second one should have automatically started. This is a good opportunity to queue up the next sound. Repeat step two.
5. Make sure the framerate is high enough to queue up a new sound before the other one finishes otherwise you will get gaps.

These sounds are not limited by format, channel count or bit depth and can also be modified to allow for overlap, by reducing the *delay* from the first sound playing to the second by the overlap amount.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to play a string of sounds together without gaps,
2  using `delay`, to produce a granular synthesis style trick engine effect.
3  """
4
5  import curses
6  import random
7  import sys
8  import time
9
10 import pyfmodex
11 from pyfmodex.enums import RESULT, TIMEUNIT
12 from pyfmodex.exceptions import FmodError
13 from pyfmodex.flags import MODE
14 from pyfmodex.structobject import Structobject
15
16 MIN_FMOD_VERSION = 0x00020108
17
18 soundnames = (
19     "media/granular/truck_idle_off_01.wav",
20     "media/granular/truck_idle_off_02.wav",
21     "media/granular/truck_idle_off_03.wav",
22     "media/granular/truck_idle_off_04.wav",
23     "media/granular/truck_idle_off_05.wav",
24     "media/granular/truck_idle_off_06.wav",
25 )
26
27 # Create a System object and initialize.
28 system = pyfmodex.System()
29 VERSION = system.version
30 if VERSION < MIN_FMOD_VERSION:
31     print(
32         f"FMOD lib version {VERSION:#08x} doesn't meet "
33         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
34     )
35     sys.exit(1)
36
37 system.init(maxchannels=2)
38
39 output_rate = system.software_format.sample_rate
40 dsp_block_len = system.dsp_buffer_size.size

```

(continues on next page)

```
41 master_channel_group = system.master_channel_group
42
43 sounds = []
44 for soundname in soundnames:
45     sounds.append(system.create_sound(soundname, mode=MODE.IGNORETAGS))
46
47
48 def queue_next_sound(playingchannel=None):
49     """Queue the next sound."""
50     newsound = sounds[random.randrange(0, len(sounds))]
51     newchannel = system.play_sound(newsound, paused=True)
52
53     start_delay = 0
54     if playingchannel:
55         # Get the start time of the playing channel
56         start_delay = playingchannel.delay.dsp_start
57
58         # Grab the length of the playing sound, and its frequency, so we can
59         # calculate where to place the new sound on the time line
60         sound_len = playingchannel.current_sound.get_length(TIMEUNIT.PCM)
61         freq = playingchannel.frequency
62
63         # Now calculate the length of the sound in 'output samples'. For
64         # instance, if a 44khz sound is 22050 samples long, and the output
65         # rate is 48khz, then we want to delay by 24000 output samples
66         sound_len = int(sound_len / freq * output_rate)
67
68         # Add output rate adjusted sound length to the clock value of the
69         # sound that is currently playing
70         start_delay += sound_len
71     else:
72         start_delay = newchannel.dsp_clock.parent_clock
73         start_delay += 2 * dsp_block_len
74
75     # Set the delay of the new sound to the end of the old sound
76     newchannel.delay = Structobject(
77         dsp_start=start_delay, dsp_end=0, stop_channels=False
78     )
79
80     # Randomize pitch/volume to make it sound more realistic / random
81     newchannel.frequency *= (
82         1 + random.uniform(-1, 1) * .02
83     ) # @22khz, range fluctuates from 21509 to 22491
84
85     newchannel.volume *= 1 - random.random() * 0.2 # 0.8 to 1.0
86
87     newchannel.paused = False
88
89     return newchannel
90
91
92 # Main loop
```

(continues on next page)

(continued from previous page)

```

93 def main(stdscr):
94     """Draw a simple TUI, grab keypresses and let the user manipulate the
95     channel paused state.
96     """
97     stdscr.clear()
98     stdscr.nodelay(True)
99
100    # Create small visual display
101    stdscr.addstr(
102        "=====\n"
103        "Granular Synthesis SetDelay Example.\n"
104        "=====\n"
105        "\n"
106        "Press SPACE to toggle pause\n"
107        "Press q to quit"
108    )
109
110    # Kick off the first two sounds. First one is immediate, second one will be
111    # triggered to start after the first one.
112    channels = []
113    channels.append(queue_next_sound())
114    channels.append(queue_next_sound(channels[0]))
115
116    slot = 0
117    while True:
118        paused_state = "paused" if master_channel_group.paused else "playing"
119
120        stdscr.move(7, 0)
121        stdscr.clrtoeol()
122        stdscr.addstr(f"Channels are {paused_state}")
123
124        # Replace the sound that just finished with a new sound, to create
125        # endless seamless stitching!
126        try:
127            is_playing = channels[slot].is_playing
128        except FmodError as fmoderror:
129            if fmoderror.result != RESULT.INVALID_HANDLE:
130                raise fmoderror
131
132        if not is_playing and not master_channel_group.paused:
133            # Replace sound that just ended with a new sound, queued up to
134            # trigger exactly after the other sound ends
135            channels[slot] = queue_next_sound(channels[1 - slot])
136            slot = 1 - slot # flip
137
138        # Listen to the user
139        try:
140            keypress = stdscr.getkey()
141            if keypress == " ":
142                master_channel_group.paused = not master_channel_group.paused
143            elif keypress == "q":
144                break

```

(continues on next page)

(continued from previous page)

```

145     except curses.error as cerr:
146         if cerr.args[0] != "no input":
147             raise cerr
148
149     system.update()
150     # If you wait too long (longer than the length of the shortest sound),
151     # you will get gaps.
152     time.sleep(10 / 1000)
153
154
155 curses.wrapper(main)
156
157 # Shut down
158 for sound in sounds:
159     sound.release()
160 system.release()

```

4.10 Load from memory

This is a sample script showing how to use the OPENMEMORY mode flag when creating sounds to load the data into memory.

This example is simply a variant of the *Play sound* example, but it loads the data into memory and then uses the *load from memory* feature of `create_sound()`.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to load data into memory and read it from there."""
2
3  import curses
4  import mmap
5  import sys
6  import time
7  from pathlib import Path
8
9  import pyfmodex
10 from pyfmodex.enums import RESULT, TIMEUNIT
11 from pyfmodex.exceptions import FmodError
12 from pyfmodex.flags import MODE
13 from pyfmodex.structures import CREATESOUNDEXINFO
14
15 MIN_FMOD_VERSION = 0x00020108
16
17 mediadir = Path("media")
18 soundnames = (
19     mediadir / "drumloop.wav",
20     mediadir / "jaguar.wav",
21     mediadir / "swish.wav",
22 )
23
24 # Create a System object and initialize

```

(continues on next page)

(continued from previous page)

```

25 system = pyfmodex.System()
26 VERSION = system.version
27 if VERSION < MIN_FMOD_VERSION:
28     print(
29         f"FMOD lib version {VERSION:#08x} doesn't meet "
30         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
31     )
32     sys.exit(1)
33
34 system.init()
35
36 sounds = []
37 for filename in soundnames:
38     with open(filename, mode="rb") as file_obj:
39         with mmap.mmap(
40             file_obj.fileno(), length=0, access=mmap.ACCESS_READ
41         ) as mmap_obj:
42             sounds.append(
43                 system.create_sound(
44                     mmap_obj.read(),
45                     mode=MODE.OPENMEMORY | MODE.LOOP_OFF,
46                     exinfo=CREATESOUNDEXINFO(length=mmap_obj.size()),
47                 )
48             )
49
50 # Main loop
51 def main(stdscr):
52     """Draw a simple TUI, grab keypresses and let the user play the sounds."""
53     stdscr.clear()
54     stdscr.nodelay(True)
55
56     # Create small visual display
57     stdscr.addstr(
58         "=====\n"
59         "Load From Memory Example.\n"
60         "=====\n"
61         "\n"
62         f"Press 1 to play a mono sound ({soundnames[0].stem})\n"
63         f"Press 2 to play a mono sound ({soundnames[1].stem})\n"
64         f"Press 3 to play a stereo sound ({soundnames[2].stem})\n"
65         "Press q to quit"
66     )
67
68     channel = None
69     currentsound = None
70     while True:
71         is_playing = False
72         position = 0
73         length = 0
74         if channel:
75             try:
76                 is_playing = channel.is_playing

```

(continues on next page)

```
77         position = channel.get_position(TIMEUNIT.MS)
78         currentsound = channel.current_sound
79         if currentsound:
80             length = currentsound.get_length(TIMEUNIT.MS)
81
82     except FmodError as fmoderror:
83         if fmoderror.result not in (
84             RESULT.INVALID_HANDLE,
85             RESULT.CHANNEL_STOLEN,
86         ):
87             raise fmoderror
88
89     stdscr.move(9, 0)
90     stdscr.clrtoeol()
91     stdscr.addstr(
92         "Time %02d:%02d:%02d/%02d:%02d:%02d : %s"
93         % (
94             position / 1000 / 60,
95             position / 1000 % 60,
96             position / 10 % 100,
97             length / 1000 / 60,
98             length / 1000 % 60,
99             length / 10 % 100,
100            "Playing" if is_playing else "Stopped",
101        ),
102    )
103    stdscr.addstr(10, 0, f"Channel Playing {system.channels_playing.channels:-2d}")
104
105    # Listen to the user
106    try:
107        keypress = stdscr.getkey()
108        if keypress in "123":
109            channel = system.play_sound(sounds[int(keypress) - 1])
110        elif keypress == "q":
111            break
112    except curses.error as cerr:
113        if cerr.args[0] != "no input":
114            raise cerr
115
116    system.update()
117    time.sleep(50 / 1000)
118
119
120    curses.wrapper(main)
121
122    # Shut down
123    for sound in sounds:
124        sound.release()
125    system.release()
```

4.11 Multiple speakers

This is a sample script showing how to play sounds on multiple speakers, and also how to assign sound subchannels (like in stereo sound) to different, individual speakers.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to play sounds on multiple speakers."""
2
3  import curses
4  import sys
5  import time
6
7  import pyfmodex
8  from pyfmodex.enums import RESULT, SPEAKERMODE, TIMEUNIT
9  from pyfmodex.exceptions import FmodError
10 from pyfmodex.flags import MODE
11
12 MIN_FMOD_VERSION = 0x00020108
13
14 CHOICES = (
15     "Mono from front left speaker",
16     "Mono from front right speaker",
17     "Mono from center speaker",
18     "Mono from surround left speaker",
19     "Mono from surround right speaker",
20     "Mono from rear left speaker",
21     "Mono from rear right speaker",
22     "Stereo from front speakers",
23     "Stereo from front speakers (channels swapped)",
24     "Stereo (right only) from center speaker",
25 )
26
27 # Create a System object and initialize
28 system = pyfmodex.System()
29 VERSION = system.version
30 if VERSION < MIN_FMOD_VERSION:
31     print(
32         f"FMOD lib version {VERSION:#08x} doesn't meet "
33         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
34     )
35     sys.exit(1)
36
37 system.init(maxchannels=len(CHOICES))
38
39 speaker_mode = SPEAKERMODE(system.software_format.speaker_mode)
40
41 sound_mono = system.create_sound("media/drumloop.wav", mode=MODE.TWOD | MODE.LOOP_OFF)
42 sound_stereo = system.create_sound("media/stereo.ogg", mode=MODE.TWOD | MODE.LOOP_OFF)
43
44
45 def is_choice_available(choice_idx):
46     """Is the given configuration choice available in the current speakermode?"""

```

(continues on next page)

```
47 if speaker_mode in (SPEAKERMODE.MONO, SPEAKERMODE.STEREO):
48     return choice_idx not in (2, 3, 4, 5, 6, 9)
49 if speaker_mode == SPEAKERMODE.QUAD:
50     return choice_idx not in (2, 5, 6, 9)
51 if speaker_mode in (SPEAKERMODE.SURROUND, SPEAKERMODE.FIVEPOINTONE):
52     return choice_idx not in (5, 6)
53
54 return True
55
56
57 def play_sound(choice_idx):
58     """Play a sound in the given configuration choice.
59
60     Returns the created channel.
61     """
62     channel = None
63     if choice_idx == 0: # Mono front left
64         channel = system.play_sound(sound_mono, paused=True)
65         channel.set_mix_levels_output(1, 0, 0, 0, 0, 0, 0, 0)
66         channel.paused = False
67     elif choice_idx == 1: # Mono front right
68         channel = system.play_sound(sound_mono, paused=True)
69         channel.set_mix_levels_output(0, 1, 0, 0, 0, 0, 0, 0)
70         channel.paused = False
71     elif choice_idx == 2: # Mono centre
72         channel = system.play_sound(sound_mono, paused=True)
73         channel.set_mix_levels_output(0, 0, 1, 0, 0, 0, 0, 0)
74         channel.paused = False
75     elif choice_idx == 3: # Mono surround left
76         channel = system.play_sound(sound_mono, paused=True)
77         channel.set_mix_levels_output(0, 0, 0, 0, 1, 0, 0, 0)
78         channel.paused = False
79     elif choice_idx == 4: # Mono surround right
80         channel = system.play_sound(sound_mono, paused=True)
81         channel.set_mix_levels_output(0, 0, 0, 0, 0, 1, 0, 0)
82         channel.paused = False
83     elif choice_idx == 5: # Mono read left
84         channel = system.play_sound(sound_mono, paused=True)
85         channel.set_mix_levels_output(0, 0, 0, 0, 0, 0, 1, 0)
86         channel.paused = False
87     elif choice_idx == 6: # Mono read right
88         channel = system.play_sound(sound_mono, paused=True)
89         channel.set_mix_levels_output(0, 0, 0, 0, 0, 0, 0, 1)
90         channel.paused = False
91     elif choice_idx == 7: # Stereo format
92         channel = system.play_sound(sound_stereo)
93     elif choice_idx == 8: # Stereo front channel swapped
94         matrix = [0, 1,
95                  1, 0]
96         channel = system.play_sound(sound_stereo, paused=True)
97         channel.set_mix_matrix(matrix, 2, 2)
98         channel.paused = False
```

(continues on next page)

(continued from previous page)

```

99     elif choice_idx == 8: # Stereo (right only) center
100         matrix = [0, 0,
101                  0, 0,
102                  0, 1]
103         channel = system.play_sound(sound_stereo, paused=True)
104         channel.set_mix_matrix(matrix, 3, 2)
105         channel.paused = False
106     return channel
107
108
109 # Main loop
110 def main(stdscr):
111     """Draw a simple TUI, grab keypresses and let the user play the sounds."""
112     stdscr.clear()
113     stdscr.nodelay(True)
114
115     # Create small visual display
116     all_opts = speaker_mode.value >= SPEAKERMODE.SEVENPOINTONE.value
117     stdscr.addstr(
118         "=====\n"
119         "Multiple Speaker Example.\n"
120         "=====\n"
121         "\n"
122         "Press j or k to select mode\n"
123         "Press SPACE to play the sound\n"
124         "Press q to quit\n"
125         "\n"
126         f"Speaker mode is set to {speaker_mode.name}"
127         " causing some speaker options to be unavailale"
128         if not all_opts
129         else ""
130     )
131
132     channel = None
133     currentsound = None
134     choice_idx = 0
135     while True:
136         stdscr.move(10, 0)
137         for idx, choice in enumerate(CHOICES):
138             available = is_choice_available(idx)
139             sel = "-" if not available else "X" if choice_idx == idx else " "
140             stdscr.addstr(f"[{sel}] {choice}\n")
141
142         is_playing = False
143         position = 0
144         length = 0
145         if channel:
146             try:
147                 is_playing = channel.is_playing
148                 position = channel.get_position(TIMEUNIT.MS)
149                 currentsound = channel.current_sound
150                 if currentsound:

```

(continues on next page)

```

151         length = currentsound.get_length(TIMEUNIT.MS)
152
153     except FmodError as fmoderror:
154         if fmoderror.result not in (
155             RESULT.INVALID_HANDLE,
156             RESULT.CHANNEL_STOLEN,
157         ):
158             raise fmoderror
159
160     stdscr.move(11 + len(CHOICES), 0)
161     stdscr.clrtoeol()
162     stdscr.addstr(
163         "Time %02d:%02d:%02d/%02d:%02d:%02d : %s\n"
164         % (
165             position / 1000 / 60,
166             position / 1000 % 60,
167             position / 10 % 100,
168             length / 1000 / 60,
169             length / 1000 % 60,
170             length / 10 % 100,
171             "Playing" if is_playing else "Stopped",
172         ),
173     )
174     stdscr.addstr(f"Channels playing: {system.channels_playing.channels:-2d}")
175
176     # Listen to the user
177     try:
178         keypress = stdscr.getkey()
179         if keypress == "k":
180             old_idx = choice_idx
181             while True:
182                 choice_idx = max(choice_idx - 1, 0)
183                 if is_choice_available(choice_idx):
184                     break
185                 if choice_idx == 0:
186                     choice_idx = old_idx
187                     break
188             elif keypress == "j":
189                 old_idx = choice_idx
190                 while True:
191                     choice_idx = min(choice_idx + 1, len(CHOICES) - 1)
192                     if is_choice_available(choice_idx):
193                         break
194                     if choice_idx == len(CHOICES) - 1:
195                         choice_idx = old_idx
196                         break
197             elif keypress == " ":
198                 channel = play_sound(choice_idx)
199             elif keypress == "q":
200                 break
201     except curses.error as cerr:
202         if cerr.args[0] != "no input":

```

(continues on next page)

(continued from previous page)

```

203         raise cerr
204
205     system.update()
206     time.sleep(50 / 1000)
207
208
209 curses.wrapper(main)
210
211 # Shut down
212 sound_mono.release()
213 sound_stereo.release()
214 system.release()

```

4.12 Multiple systems

This example shows how to play sounds on two different output devices from the same application. It creates two System objects, selects a different sound device for each, then allows the user to play one sound on each device.

Note that sounds created on device A cannot be played on device B and vice versa.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to play sounds on two diferent output devices from
2  the same application.
3  """
4
5  import curses
6  import sys
7  import time
8
9  import pyfmodex
10 from pyfmodex.enums import OUTPUTTYPE
11 from pyfmodex.flags import MODE
12
13 MIN_FMOD_VERSION = 0x00020108
14
15
16 def fetch_driver(stdscr, system, name=""):
17     """Draw a simple TUI, grab keypresses and let the user select a driver."""
18     num_drivers = system.num_drivers
19     if not num_drivers:
20         system.output = OUTPUTTYPE.NOSOUND
21         return 0
22
23     selected_idx = 0
24     drivers = [system.get_driver_info(idx).name.decode() for idx in range(num_drivers)]
25     while True:
26         stdscr.addstr(4, 0, "Choose a device for system ")
27         stdscr.addstr(name, curses.A_BOLD)
28         stdscr.addstr(
29             "\n"

```

(continues on next page)

```

30     "\n"
31     "Use j and k to select\n"
32     "Press SPACE to confirm\n"
33     "\n"
34 )
35
36 for idx in range(num_drivers):
37     sel = "X" if selected_idx == idx else " "
38     stdscr.addstr(f"[{sel}] - {idx}. {drivers[idx]}\n")
39
40 # Listen to the user
41 try:
42     keypress = stdscr.getkey()
43     if keypress == "k":
44         selected_idx = max(selected_idx - 1, 0)
45     elif keypress == "j":
46         selected_idx = min(selected_idx + 1, num_drivers - 1)
47     elif keypress == " ":
48         return selected_idx
49 except curses.error as cerr:
50     if cerr.args[0] != "no input":
51         raise cerr
52
53     time.sleep(50 / 1000)
54
55 # Main loop
56 def main(stdscr):
57     """Draw a simple TUI, grab keypresses and let the user play some sounds."""
58     stdscr.clear()
59     stdscr.nodelay(True)
60
61     # Create small visual display
62     stdscr.addstr(
63         "=====\n"
64         "Multiple System Example.\n"
65         "====="
66     )
67 )
68
69 # Create Sound Card A
70 system_a = pyfmodex.System()
71 version = system_a.version
72 if version < MIN_FMOD_VERSION:
73     print(
74         f"FMOD lib version {version:#08x} doesn't meet "
75         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
76     )
77     sys.exit(1)
78
79 system_a.driver = fetch_driver(stdscr, system_a, "System A")
80 system_a.init()
81

```

(continues on next page)

(continued from previous page)

```

82  # Create Sound Card B
83  system_b = pyfmodex.System()
84  system_b.driver = fetch_driver(stdscr, system_b, "System B")
85  system_b.init()
86
87  # Load one sample into each sound card
88  sound_a = system_a.create_sound("media/drumloop.wav", mode=MODE.LOOP_OFF)
89  sound_b = system_b.create_sound("media/jaguar.wav")
90
91  stdscr.move(4, 0)
92  stdscr.clrtoobot()
93  stdscr.addstr(
94      "Press 1 to play a sound on device A\n"
95      "Press 2 to play a sound on device B\n"
96      "Press q to quit"
97  )
98  while True:
99      stdscr.move(8, 0)
100     stdscr.clrtoobot()
101     stdscr.addstr(
102         f"Channels playing on A: {system_a.channels_playing.channels}\n"
103         f"Channels playing on B: {system_b.channels_playing.channels}"
104     )
105
106     # Listen to the user
107     try:
108         keypress = stdscr.getkey()
109         if keypress == "1":
110             system_a.play_sound(sound_a)
111         elif keypress == "2":
112             system_b.play_sound(sound_b)
113         elif keypress == "q":
114             break
115     except curses.error as cerr:
116         if cerr.args[0] != "no input":
117             raise cerr
118
119     system_a.update()
120     system_b.update()
121     time.sleep(50 / 1000)
122
123     # Shut down
124     sound_a.release()
125     system_a.release()
126
127     sound_b.release()
128     system_b.release()
129
130
131 curses.wrapper(main)

```

4.13 Net stream

This example shows how to play streaming audio from an Internet source.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to play streaming audio from an Internet source."""
2
3  import ctypes
4  import curses
5  import sys
6  import time
7
8  import pyfmodex
9  from pyfmodex.enums import OPENSTATE, RESULT, TAGDATATYPE, TAGTYPE, TIMEUNIT
10 from pyfmodex.exceptions import FmodError
11 from pyfmodex.flags import MODE
12 from pyfmodex.structobject import Structobject
13 from pyfmodex.structures import CREATESOUNDEXINFO
14
15 URL = "https://focus.stream.publicradio.org/focus.mp3"
16
17 MIN_FMOD_VERSION = 0x00020108
18
19 # Create a System object and initialize
20 system = pyfmodex.System()
21 VERSION = system.version
22 if VERSION < MIN_FMOD_VERSION:
23     print(
24         f"FMOD lib version {VERSION:#08x} doesn't meet "
25         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
26     )
27     sys.exit(1)
28
29 system.init(maxchannels=1)
30
31 # Increase the file buffer size a little bit to account for Internet lag
32 system.stream_buffer_size = Structobject(size=64 * 1024, unit=TIMEUNIT.RAWBYTES)
33
34 # Increase the default file chunk size to handle seeking inside large playlist
35 # files that may be over 2kb.
36 exinfo = CREATESOUNDEXINFO(filebuffersize=1024 * 16)
37
38 tags = {}
39
40
41 def show_tags(stdscr, sound, channel):
42     """Read and print any tags that have arrived. This could, for example,
43     happen if a radio station switches to a new song.
44     """
45     stdscr.move(11, 0)
46     stdscr.addstr("Tags:\n")
47     while True:

```

(continues on next page)

(continued from previous page)

```

48     try:
49         tag = sound.get_tag(-1)
50     except FmodError:
51         break
52
53     if tag.datatype == TAGDATATYPE.STRING.value:
54         tag_data = ctypes.string_at(tag.data).decode()
55         tags[tag.name.decode()] = (tag_data, tag.datalen)
56         if tag.type == TAGTYPE.PLAYLIST.value and not tag.name == "FILE":
57             # data point to sound owned memory, copy it before the
58             # sound is released
59             sound.release()
60             sound = system.create_sound(
61                 tag.data,
62                 mode=MODE.CREATESTREAM | MODE.NONBLOCKING,
63                 exinfo=exinfo,
64             )
65     elif tag.type == TAGTYPE.FMOD.value:
66         # When a song changes, the sample rate may also change, so
67         # compensate here
68         if tag.name.decode() == "Sample Rate Change" and channel:
69             channel.frequency = float(ctypes.string_at(tag.data).decode())
70
71     stdscr.move(12, 0)
72     stdscr.clrtoeb()
73     for name, value in tags.items():
74         stdscr.addstr(f"{name} = '{value[0]}' ({value[1]} bytes)\n")
75
76
77 # Main loop
78 def main(stdscr):
79     """Draw a simple TUI, grab keypresses and let the user control playback."""
80     stdscr.clear()
81     stdscr.nodelay(True)
82
83     # Create small visual display
84     stdscr.addstr(
85         "=====\n"
86         "Net Stream Example.\n"
87         "=====\n"
88         "\n"
89         "Press SPACE to toggle pause\n"
90         "Press q to quit\n"
91     )
92
93     sound = system.create_sound(
94         URL, mode=MODE.CREATESTREAM | MODE.NONBLOCKING, exinfo=exinfo
95     )
96
97     channel = None
98     while True:
99         open_state = sound.open_state

```

(continues on next page)

```
100
101     is_playing = False
102     position = 0
103     paused = False
104     if channel:
105         try:
106             is_playing = channel.is_playing
107             position = channel.get_position(TIMEUNIT.MS)
108             paused = channel.paused
109
110             # Silence the stream until we have sufficient data for smooth
111             # playback
112             channel.mute = open_state.starving
113         except FmodError as fmoderror:
114             if fmoderror.result not in (
115                 RESULT.INVALID_HANDLE,
116                 RESULT.CHANNEL_STOLEN,
117             ):
118                 raise fmoderror
119     else:
120         try:
121             channel = system.play_sound(sound)
122         except FmodError:
123             # This may fail if the stream isn't ready yet, so don't check
124             # for errors
125             pass
126
127     state = ""
128     if open_state.state == OPENSTATE.BUFFERING:
129         state = "Buffering..."
130     elif open_state.state == OPENSTATE.CONNECTING:
131         state = "Connecting..."
132     elif paused:
133         state = "Paused"
134     elif is_playing:
135         state = "Playing"
136
137     if open_state.starving:
138         state += " (STARVING)"
139
140     stdscr.move(7, 0)
141     stdscr.clrtoeol()
142     stdscr.addstr(
143         "Time = %02d:%02d:%02d\n"
144         % (
145             position / 1000 / 60,
146             position / 1000 % 60,
147             position / 10 % 100,
148         ),
149     )
150     stdscr.addstr(
151         f"State = {state}\n"
```

(continues on next page)

(continued from previous page)

```

152         f"Buffer Percentage = {open_state.percent_buffered}%"
153     )
154
155     show_tags(stdscr, sound, channel)
156
157     # Listen to the user
158     try:
159         keypress = stdscr.getkey()
160         if keypress == " ":
161             if channel:
162                 channel.paused = not channel.paused
163             elif keypress == "q":
164                 break
165         except curses.error as cerr:
166             if cerr.args[0] != "no input":
167                 raise cerr
168
169         system.update()
170         time.sleep(50 / 1000)
171
172     if channel:
173         channel.stop()
174
175     stdscr.clear()
176     stdscr.addstr("Waiting for sound to finish opening before trying to release it...")
177     stdscr.refresh()
178     while True:
179         if sound.open_state.state == OPENSTATE.READY:
180             break
181         system.update()
182         time.sleep(50 / 1000)
183
184     sound.release()
185
186
187 curses.wrapper(main)
188
189 # Shut down
190 system.release()

```

4.14 Play sound

This example shows how to simply load and play multiple sounds, the simplest usage of FMOD. By default FMOD will decode the entire file into memory when it loads. If the sounds are big and possibly take up a lot of RAM it would be better to use the CREATESTREAM flag, as this will stream the file in realtime as it plays (see *Play stream*).

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to simply load and play multiple sounds, the
2  simplest usage of FMOD.

```

(continues on next page)

```
3  """
4
5  import curses
6  import sys
7  import time
8
9  from pathlib import Path
10
11  import pyfmodex
12  from pyfmodex.enums import RESULT, TIMEUNIT
13  from pyfmodex.exceptions import FmodError
14  from pyfmodex.flags import MODE
15
16  MIN_FMOD_VERSION = 0x00020108
17
18  mediadir = Path("media")
19  soundnames = (
20      mediadir / "drumloop.wav",
21      mediadir / "jaguar.wav",
22      mediadir / "swish.wav",
23  )
24
25  # Create a System object and initialize
26  system = pyfmodex.System()
27  VERSION = system.version
28  if VERSION < MIN_FMOD_VERSION:
29      print(
30          f"FMOD lib version {VERSION:#08x} doesn't meet "
31          f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
32      )
33      sys.exit(1)
34
35  system.init()
36
37  sounds = []
38  for filename in soundnames:
39      # drumloop.wav has embedded loop points which automatically turns on
40      # looping so we turn it off (for all) here.
41      sounds.append(system.create_sound(str(filename), mode=MODE.LOOP_OFF))
42
43
44  # Main loop
45  def main(stdscr):
46      """Draw a simple TUI, grab keypresses and let the user play the sounds."""
47      stdscr.clear()
48      stdscr.nodelay(True)
49
50      # Create small visual display
51      stdscr.addstr(
52          "=====\n"
53          "Play Sound Example.\n"
54          "=====\n"
```

(continues on next page)

(continued from previous page)

```

55     "\n"
56     f"Press 1 to play a mono sound ({soundnames[0].stem})\n"
57     f"Press 2 to play a mono sound ({soundnames[1].stem})\n"
58     f"Press 3 to play a stereo sound ({soundnames[2].stem})\n"
59     "Press q to quit"
60 )
61
62 channel = None
63 currentsound = None
64 while True:
65     is_playing = False
66     position = 0
67     length = 0
68     if channel:
69         try:
70             is_playing = channel.is_playing
71             position = channel.get_position(TIMEUNIT.MS)
72             currentsound = channel.current_sound
73             if currentsound:
74                 length = currentsound.get_length(TIMEUNIT.MS)
75
76         except FmodError as fmoderror:
77             if fmoderror.result not in (
78                 RESULT.INVALID_HANDLE,
79                 RESULT.CHANNEL_STOLEN,
80             ):
81                 raise fmoderror
82
83     stdscr.move(9, 0)
84     stdscr.clrtoeol()
85     stdscr.addstr(
86         "Time %02d:%02d:%02d/%02d:%02d:%02d : %s"
87         % (
88             position / 1000 / 60,
89             position / 1000 % 60,
90             position / 10 % 100,
91             length / 1000 / 60,
92             length / 1000 % 60,
93             length / 10 % 100,
94             "Playing" if is_playing else "Stopped",
95         ),
96     )
97     stdscr.addstr(10, 0, f"Channel Playing {system.channels_playing.channels:-2d}")
98
99     # Listen to the user
100    try:
101        keypress = stdscr.getkey()
102        if keypress in "123":
103            channel = system.play_sound(sounds[int(keypress) - 1])
104        elif keypress == "q":
105            break
106    except curses.error as cerr:

```

(continues on next page)

(continued from previous page)

```

107         if cerr.args[0] != "no input":
108             raise cerr
109
110         system.update()
111         time.sleep(50 / 1000)
112
113
114 curses.wrapper(main)
115
116 # Shut down
117 for sound in sounds:
118     sound.release()
119 system.release()

```

4.15 Play stream

This example shows how to simply play a stream such as an MP3 or WAV. The stream behaviour is achieved by specifying `CREATESTREAM` in the call to `create_sound()`. This makes FMOD decode the file in realtime as it plays, instead of loading it all at once which uses far less memory in exchange for a small runtime CPU hit.

Note that *pyfmodex* does this automatically through the convenience method `create_stream()`.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to simply play a stream such as an MP3 or WAV."""
2
3  import curses
4  import sys
5  import time
6  from pathlib import Path
7
8  import pyfmodex
9  from pyfmodex.enums import RESULT, TIMEUNIT
10 from pyfmodex.exceptions import FmodError
11 from pyfmodex.flags import MODE
12
13 MIN_FMOD_VERSION = 0x00020108
14
15 mediadir = Path("media")
16 soundnames = (
17     mediadir / "drumloop.wav",
18     mediadir / "jaguar.wav",
19     mediadir / "swish.wav",
20 )
21
22 # Create a System object and initialize
23 system = pyfmodex.System()
24 VERSION = system.version
25 if VERSION < MIN_FMOD_VERSION:
26     print(
27         f"FMOD lib version {VERSION:#08x} doesn't meet "

```

(continues on next page)

(continued from previous page)

```

28     f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
29 )
30 sys.exit(1)
31
32 system.init()
33
34 # This example uses an FSB file, which is a preferred pack format for fmod
35 # containing multiple sounds. This could just as easily be exchanged with a
36 # wav/mp3/ogg file for example, but in that case you wouldn't need to check for
37 # subsounds. Because of the check below, this example would work with both
38 # types of sound file (packed vs single).
39 sound = system.create_stream("media/wave_vorbis.fsb", mode=MODE.LOOP_NORMAL)
40
41 sound_to_play = sound
42 if sound.num_subsounds:
43     sound_to_play = sound.get_subsound(0)
44
45 # Main loop
46 def main(stdscr):
47     """Draw a simple TUI, grab keypresses and let the user control playback."""
48     stdscr.clear()
49     stdscr.nodelay(True)
50
51     # Create small visual display
52     stdscr.addstr(
53         "=====\n"
54         "Play Stream Example.\n"
55         "=====\n"
56         "\n"
57         "Press SPACE to toggle pause\n"
58         "Press q to quit"
59     )
60
61     # Play the sound
62     channel = sound_to_play.play()
63
64     while True:
65         is_playing = False
66         position = 0
67         length = 0
68         try:
69             is_playing = channel.is_playing
70             position = channel.get_position(TIMEUNIT.MS)
71             length = sound_to_play.get_length(TIMEUNIT.MS)
72         except FmodError as fmoderror:
73             if not fmoderror.result is RESULT.INVALID_HANDLE:
74                 raise fmoderror
75
76         stdscr.move(7, 0)
77         stdscr.clrtoeol()
78         stdscr.addstr(
79             "Time %02d:%02d:%02d/%02d:%02d:%02d : %s"

```

(continues on next page)

```

80         % (
81             position / 1000 / 60,
82             position / 1000 % 60,
83             position / 10 % 100,
84             length / 1000 / 60,
85             length / 1000 % 60,
86             length / 10 % 100,
87             "Paused" if channel.paused else "Playing" if is_playing else "Stopped",
88         ),
89     )
90
91     # Listen to the user
92     try:
93         keypress = stdscr.getkey()
94         if keypress == " ":
95             channel.paused = not channel.paused
96         elif keypress == "q":
97             break
98     except curses.error as cerr:
99         if cerr.args[0] != "no input":
100             raise cerr
101
102     system.update()
103     time.sleep(50 / 1000)
104
105
106 curses.wrapper(main)
107
108 # Shut down
109 sound_to_play.release()
110 system.release()

```

4.16 Record enumeration

This example shows how to enumerate the available recording drivers on a device. It demonstrates how the enumerated list changes as microphones are attached and detached. It also shows that you can record from multi mics at the same time (if your audio subsystem supports that).

Please note: to minimize latency, care should be taken to control the number of samples between the record position and the play position. Check *Record* for details on this process.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to enumerate the available recording drivers on
2  this device and work with them.
3  """
4
5  import curses
6  import sys
7  import time
8  from collections import defaultdict

```

(continues on next page)

(continued from previous page)

```

9  from ctypes import c_int, c_short, sizeof
10
11 import pyfmodex
12 from pyfmodex.enums import RESULT, SOUND_FORMAT
13 from pyfmodex.exceptions import FmodError
14 from pyfmodex.flags import DRIVER_STATE, MODE, SYSTEM_CALLBACK_TYPE
15 from pyfmodex.structures import CREATESOUNDEXINFO
16
17 MIN_FMOD_VERSION = 0x00020108
18 MAX_DRIVERS_IN_VIEW = 3
19 MAX_DRIVERS = 16
20
21 # Create a System object and initialize
22 system = pyfmodex.System()
23 VERSION = system.version
24 if VERSION < MIN_FMOD_VERSION:
25     print(
26         f"FMOD lib version {VERSION:#08x} doesn't meet "
27         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
28     )
29     sys.exit(1)
30
31 system.init()
32
33 # Setup a callback so we can be notified if the record list has changed
34 def record_list_changed_callback( # pylint: disable=unused-argument
35     mysystem, callback_type, commanddata1, comanddata2, userdata
36 ):
37     """Increase a counter referenced by userdata."""
38     _record_list_changed_count = c_int.from_address(userdata)
39     _record_list_changed_count.value += 1
40
41     return RESULT.OK.value
42
43
44 record_list_changed_count = c_int(0)
45 system.user_data = record_list_changed_count
46 system.set_callback(
47     record_list_changed_callback, SYSTEM_CALLBACK_TYPE.RECORDLISTCHANGED
48 )
49
50
51 recordings = [defaultdict(bool) for _ in range(MAX_DRIVERS)]
52
53
54 def show_record_drivers(stdscr, selected_driver_idx, num_drivers):
55     """Show an overview of detected record drivers."""
56     row, _ = stdscr.getyx()
57     for i in range(min(MAX_DRIVERS_IN_VIEW, num_drivers)):
58         idx = (selected_driver_idx - MAX_DRIVERS_IN_VIEW // 2 + i) % num_drivers
59         row += 2
60         if idx == selected_driver_idx:

```

(continues on next page)

```

61         stdscr.addstr(row, 0, ">")
62         driver_info = system.get_record_driver_info(idx)
63         statechar = "(*) " if DRIVER_STATE(driver_info.state) & DRIVER_STATE.DEFAULT_
↪ else ""
64         stdscr.addstr(row, 2, f"{idx}. {statechar}{driver_info.name.decode():41s}")
65         row += 1
66         stdscr.addstr(row, 2, f"{driver_info.system_rate/1000:2.1f}KHz")
67         stdscr.addstr(row, 10, f"{driver_info.speaker_mode_channels}ch")
68         data4 = driver_info.guid.data4.zfill(8).decode()
69         stdscr.addstr(
70             row,
71             13,
72             "{%08X-%04X-%04X-%04X-%02X%02X%02X%02X%02X%02X}"
73             % (
74                 driver_info.guid.data1,
75                 driver_info.guid.data2,
76                 driver_info.guid.data3,
77                 int(data4[0]) << 8 | int(data4[1]),
78                 int(data4[2]),
79                 int(data4[3]),
80                 int(data4[4]),
81                 int(data4[5]),
82                 int(data4[6]),
83                 int(data4[7]),
84             ),
85         )
86         row += 1
87         stdscr.addstr(
88             row,
89             2,
90             "(%s) (%s) (%s)"
91             % (
92                 "Connected"
93                 if (DRIVER_STATE(driver_info.state) & DRIVER_STATE.CONNECTED)
94                 else "Unplugged",
95                 "Recording" if system.is_recording(idx) else "Not recoding",
96                 "Playing"
97                 if recordings[idx]["channel"] and recordings[idx]["channel"].is_playing
98                 else "Not playing",
99             ),
100         )
101
102
103 # Main loop
104 def main(stdscr):
105     """Draw a simple TUI, grab keypresses and let the user control recording
106     and playback.
107     """
108     stdscr.clear()
109     stdscr.nodelay(True)
110
111     # Create small visual display

```

(continues on next page)

(continued from previous page)

```

112 stdscr.addstr(
113     "=====\n"
114     "Record Enumeration Example.\n"
115     "====="
116 )
117
118 selected_driver_idx = 0
119 cur_y, _ = stdscr.getyx()
120 while True:
121     stdscr.move(cur_y + 2, 0)
122     stdscr.clrtoobot()
123     stdscr.addstr(
124         f"Record list has updated {record_list_changed_count.value} time(s)\n"
125         f"Currently, {system.record_num_drivers.connected} recording device(s) are
↳plugged in\n"
126         "\n"
127         "Press j and k to scroll list\n"
128         "Press q to quit\n"
129         "\n"
130         "Press 1 to start/stop recording\n"
131         "Press 2 to start/stop playback"
132     )
133
134     # Clamp the reported number of drivers to simplify this example
135     num_drivers = min(system.record_num_drivers.drivers, MAX_DRIVERS)
136
137     subwin = stdscr.subwin(cur_y + 9, 0)
138     show_record_drivers(subwin, selected_driver_idx, num_drivers)
139
140     # Listen to the user
141     try:
142         keypress = stdscr.getkey()
143         if keypress == "j":
144             selected_driver_idx = (selected_driver_idx + 1) % num_drivers
145         elif keypress == "k":
146             selected_driver_idx = (selected_driver_idx - 1) % num_drivers
147         elif keypress == "q":
148             break
149         elif keypress == "1":
150             if system.is_recording(selected_driver_idx):
151                 system.record_stop(selected_driver_idx)
152             else:
153                 # Clean up previous record sound
154                 if recordings[selected_driver_idx]["sound"]:
155                     recordings[selected_driver_idx]["sound"].release()
156
157                 # Query device native settings and start a recording
158                 record_driver_info = system.get_record_driver_info(
159                     selected_driver_idx
160                 )
161                 exinfo = CREATESOUNDEXINFO(
162                     numchannels=record_driver_info.speaker_mode_channels,

```

(continues on next page)

(continued from previous page)

```

163         format=SOUND_FORMAT.PCM16.value,
164         defaultfrequency=record_driver_info.system_rate,
165         # one second buffer; size here does not change the latency
166         length=record_driver_info.system_rate
167         * sizeof(c_short)
168         * record_driver_info.speaker_mode_channels,
169     )
170     sound = system.create_sound(
171         0, mode=MODE.LOOP_NORMAL | MODE.OPENUSER, exinfo=exinfo
172     )
173     recordings[selected_driver_idx]["sound"] = sound
174     try:
175         system.record_start(selected_driver_idx, sound, loop=True)
176     except FmodError as fmoderror:
177         if fmoderror.result != RESULT.RECORD_DISCONNECTED:
178             raise fmoderror
179     elif keypress == "2":
180         channel = recordings[selected_driver_idx]["channel"]
181         sound = recordings[selected_driver_idx]["sound"]
182         if channel and channel.is_playing:
183             channel.stop()
184             recordings[selected_driver_idx]["channel"] = False
185         elif sound:
186             recordings[selected_driver_idx]["channel"] = sound.play()
187
188     except curses.error as cerr:
189         if cerr.args[0] != "no input":
190             raise cerr
191
192     system.update()
193     time.sleep(50 / 1000)
194
195
196 curses.wrapper(main)
197
198 # Shut down
199 for recorder in recordings:
200     if recorder["sound"]:
201         recorder["sound"].release()
202 system.release()

```

4.17 Record

This example shows how to record continuously and play back the same data while keeping a specified latency between the two. This is achieved by delaying the start of playback until the specified number of milliseconds has been recorded. At runtime the playback speed will be slightly altered to compensate for any drift in either play or record drivers.

(Adapted from sample code shipped with FMOD Engine.)

```
1 """Example code to show how to record continuously and play back the same data
```

(continues on next page)

(continued from previous page)

```

2  while keeping a specified latency between the two.
3  """
4
5  import curses
6  import sys
7  import time
8  from ctypes import c_short, sizeof
9
10 import pyfmodex
11 from pyfmodex.enums import RESULT, SOUND_FORMAT, TIMEUNIT
12 from pyfmodex.exceptions import FmodError
13 from pyfmodex.flags import MODE
14 from pyfmodex.reverb_presets import REVERB_PRESET
15 from pyfmodex.structures import CREATESOUNDEXINFO, REVERB_PROPERTIES
16
17 MIN_FMOD_VERSION = 0x00020108
18
19 # Some devices will require higher latency to avoid glitches
20 LATENCY_MS = 50
21 DRIFT_MS = 1
22 RECORD_DEVICE_INDEX = 0
23
24 # Create a System object and initialize
25 system = pyfmodex.System()
26 VERSION = system.version
27 if VERSION < MIN_FMOD_VERSION:
28     print(
29         f"FMOD lib version {VERSION:#08x} doesn't meet "
30         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
31     )
32     sys.exit(1)
33
34 system.init()
35
36 if not system.record_num_drivers:
37     print("No recording devices found/plugged in! Aborting.")
38     sys.exit(1)
39
40 # Determine latency in samples
41 record_driver_info = system.get_record_driver_info(RECORD_DEVICE_INDEX)
42
43 # The point where we start compensating for drift
44 drift_threshold = record_driver_info.system_rate * DRIFT_MS / 1000
45 # User specified latency
46 desired_latency = record_driver_info.system_rate * LATENCY_MS / 1000
47
48 # Create user sound to record into, then start recording
49 exinfo = CREATESOUNDEXINFO(
50     numchannels=record_driver_info.speaker_mode_channels,
51     format=SOUND_FORMAT.PCM16.value,
52     defaultfrequency=record_driver_info.system_rate,
53     # one second buffer; size here does not change the latency

```

(continues on next page)

```

54     length=record_driver_info.system_rate
55     * sizeof(c_short)
56     * record_driver_info.speaker_mode_channels,
57 )
58 sound = system.create_sound(0, mode=MODE.LOOP_NORMAL | MODE.OPENUSER, exinfo=exinfo)
59 system.record_start(RECORD_DEVICE_INDEX, sound, loop=True)
60 sound_len = sound.get_length(TIMEUNIT.PCM)
61
62 # Main loop
63 def main(stdscr):
64     """Draw a simple TUI, grab keypresses and let the user control playback."""
65     stdscr.clear()
66     stdscr.nodelay(True)
67
68     dsp_enabled = False
69
70     # Create small visual display
71     stdscr.addstr(
72         "=====\n"
73         "Record Example.\n"
74         "=====\n"
75         "\n"
76         "(Adjust LATENCY_MS in the source to compensate for stuttering)\n"
77         f"(Current value is {LATENCY_MS}ms)"
78     )
79
80     reverb_on = REVERB_PROPERTIES(*REVERB_PRESET.CONCERTHALL.value)
81     reverb_off = REVERB_PROPERTIES(*REVERB_PRESET.OFF.value)
82
83     # User specified latency adjusted for driver update granularity
84     adjusted_latency = desired_latency
85     # Latency measured once playback begins (smoothened for jitter)
86     actual_latency = desired_latency
87
88     last_record_pos = 0
89     last_play_pos = 0
90     samples_recorded = 0
91     samples_played = 0
92     min_record_delta = sound_len
93     channel = None
94     while True:
95         stdscr.move(7, 0)
96         stdscr.clrtoeol()
97         stdscr.addstr(
98             f"Press SPACE to {'disable' if dsp_enabled else 'enable'} DSP effect\n"
99             "Press q to quit"
100        )
101
102        # Determine how much has been recorded since we last checked
103        record_pos = 0
104        try:
105            record_pos = system.get_record_position(RECORD_DEVICE_INDEX)

```

(continues on next page)

(continued from previous page)

```

106     except FmodError as fmoderror:
107         if fmoderror.result != RESULT.RECORD_DISCONNECTED:
108             raise fmoderror
109
110     record_delta = (
111         record_pos - last_record_pos
112         if record_pos >= last_record_pos
113         else record_pos + sound_len - last_record_pos
114     )
115     last_record_pos = record_pos
116     samples_recorded += record_delta
117
118     if record_delta and record_delta < min_record_delta:
119         # Smallest driver granularity seen so far
120         min_record_delta = record_delta
121         # Adjust our latency if driver granularity is high
122         adjusted_latency = max(desired_latency, record_delta)
123
124     # Delay playback until our desired latency is reached
125     if not channel and samples_recorded >= adjusted_latency:
126         channel = sound.play()
127
128     if channel:
129         # Stop playback if recording stops
130         if not system.is_recording(RECORD_DEVICE_INDEX):
131             channel.paused = True
132
133         # Determine how much has been played since we last checked
134         play_pos = channel.get_position(TIMEUNIT.PCM)
135         play_delta = (
136             play_pos - last_play_pos
137             if play_pos >= last_play_pos
138             else play_pos + sound_len - last_play_pos
139         )
140         last_play_pos = play_pos
141         samples_played += play_delta
142
143         # Compensate for any drift
144         latency = samples_recorded - samples_played
145         actual_latency = 0.97 * actual_latency + 0.03 * latency
146
147         playbackrate = record_driver_info.system_rate
148         if actual_latency < adjusted_latency - drift_threshold:
149             # Play position is catching up to the record position, slow
150             # playback down by 2%
151             playbackrate -= playbackrate / 50
152         elif actual_latency > adjusted_latency + drift_threshold:
153             # Play position is falling behind the record position, speed
154             # playback up by 2%
155             playbackrate += playbackrate / 50
156         channel.frequency = playbackrate
157

```

(continues on next page)

```
158     adjusted_latency_ms = int(  
159         adjusted_latency * 1000 / record_driver_info.system_rate  
160     )  
161     actual_latency_ms = int(actual_latency * 1000 / record_driver_info.system_rate)  
162     samples_recorded_s = int(samples_recorded / record_driver_info.system_rate)  
163     samples_played_s = int(samples_played / record_driver_info.system_rate)  
164  
165     stdscr.move(10, 0)  
166     stdscr.clrtoobot()  
167     stdscr.addstr(  
168         f"Adjusted latency: {adjusted_latency:4.0f} ({adjusted_latency_ms}ms)\n"  
169         f"Actual latency:   {actual_latency:4.0f} ({actual_latency_ms}ms)\n"  
170         "\n"  
171         f"Recorded: {samples_recorded:5d} ({samples_recorded_s}s)\n"  
172         f"Played: {samples_played:5d} ({samples_played_s}s)"  
173     )  
174  
175     # Listen to the user  
176     try:  
177         keypress = stdscr.getkey()  
178         if keypress == " ":  
179             # Add a DSP effect -- just for fun  
180             dsp_enabled = not dsp_enabled  
181             system.set_reverb_properties(  
182                 0, reverb_on if dsp_enabled else reverb_off  
183             )  
184             elif keypress == "q":  
185                 break  
186     except curses.error as cerr:  
187         if cerr.args[0] != "no input":  
188             raise cerr  
189  
190     system.update()  
191     time.sleep(10 / 1000)  
192  
193  
194 curses.wrapper(main)  
195  
196 # Shut down  
197 sound.release()  
198 system.release()
```

4.18 User Created Sound

This example shows how create a sound with data filled by the user. It shows a user created static sample, followed by a user created stream. The former allocates all memory needed for the sound and is played back as a static sample, while the latter streams the data in chunks as it plays, using far less memory.

(Adapted from sample code shipped with FMOD Engine.)

```

1  """Example code to show how to create a sound with data filled by the user."""
2
3  import curses
4  import sys
5  import time
6  from ctypes import c_float, c_short, sizeof
7  from math import sin
8
9  import pyfmodex
10 from pyfmodex.callback_prototypes import (SOUND_PCMREADCALLBACK,
11                                           SOUND_PCMSETPOSCALLBACK)
12 from pyfmodex.enums import RESULT, SOUND_FORMAT, TIMEUNIT
13 from pyfmodex.exceptions import FmodError
14 from pyfmodex.flags import MODE
15 from pyfmodex.structures import CREATESOUNDEXINFO
16
17 MIN_FMOD_VERSION = 0x00020108
18
19 # Create a System object and initialize
20 system = pyfmodex.System()
21 VERSION = system.version
22 if VERSION < MIN_FMOD_VERSION:
23     print(
24         f"FMOD lib version {VERSION:#08x} doesn't meet "
25         f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
26     )
27     sys.exit(1)
28
29 system.init()
30
31 # pylint: disable=invalid-name
32 # Using names common in mathematics
33 t1, t2 = c_float(0), c_float(0) # time
34 v1, v2 = c_float(0), c_float(0) # velocity
35
36
37 def pcmread_callback(sound_p, data_p, datalen_i): # pylint: disable=unused-argument
38     """Read callback used for user created sounds.
39
40     Generates smooth noise.
41     """
42
43     # >>2 = 16bit stereo (4 bytes per sample)
44     for _ in range(datalen_i >> 2):
45         # left channel

```

(continues on next page)

```

46     stereo16bitbuffer_left = int(sin(t1.value) * 32767)
47     c_short.from_address(data_p).value = stereo16bitbuffer_left
48     data_p += sizeof(c_short)
49
50     # right channel
51     stereo16bitbuffer_right = int(sin(t2.value) * 32767)
52     c_short.from_address(data_p).value = stereo16bitbuffer_right
53     data_p += sizeof(c_short)
54
55     t1.value += 0.01 + v1.value
56     t2.value += 0.0142 + v2.value
57     v1.value += sin(t1.value) * 0.002
58     v2.value += sin(t2.value) * 0.002
59
60     return RESULT.OK.value
61
62
63 def pcmsetpos_callback(
64     sound, subsound, position, timeunit
65 ): # pylint: disable=unused-argument
66     """Set position callback for user created sounds or to intercept FMOD's
67     decoder during an API setPosition call.
68
69     This is useful if the user calls setPosition on a channel and you want to
70     seek your data accordingly.
71     """
72     return RESULT.OK.value
73
74
75 # Main loop
76 def main(stdscr):
77     """Draw a simple TUI, grab keypresses and let the user select a sound
78     generation method.
79     """
80     stdscr.clear()
81     stdscr.nodelay(True)
82
83     # Create small visual display
84     stdscr.addstr(
85         "=====\n"
86         "User Created Sound Example.\n"
87         "====="
88     )
89     stdscr.refresh()
90
91     subwin = stdscr.derwin(4, 0)
92     subwin.nodelay(True)
93     subwin.addstr(
94         "Sound played here is generated in realtime. It will either play as a "
95         "stream which means it is continually filled as it is playing, or it "
96         "will play as a static sample, which means it is filled once as the "
97         "sound is created, then, when played, it will just play that short "

```

(continues on next page)

(continued from previous page)

```

98     "loop of data.\n"
99     "\n"
100    "Press 1 to play an generated infinite stream\n"
101    "Press 2 to play a static looping sample\n"
102    "Press q to quit"
103    )
104
105    mode = MODE.OPENUSER | MODE.LOOP_NORMAL
106    while True:
107        # Listen to the user
108        try:
109            keypress = subwin.getkey()
110            if keypress == "1":
111                mode |= MODE.CREATESTREAM
112                break
113            if keypress == "2":
114                break
115            if keypress == "q":
116                return
117        except curses.error as cerr:
118            if cerr.args[0] != "no input":
119                raise cerr
120
121        time.sleep(50 / 1000)
122
123    # Create and play the sound
124    numchannels = 2
125    defaultfrequency = 44100
126    exinfo = CREATESOUNDEXINFO(
127        # Number of channels in the sound
128        numchannels=numchannels,
129        # Default playback rate of the sound
130        defaultfrequency=defaultfrequency,
131        # Chunk size of stream update in samples. This will be the amount of
132        # data passed to the user callback.
133        decodebuffersize=44100,
134        # Length of PCM data in bytes of whole sound (for sound.get_length)
135        length=defaultfrequency * numchannels * sizeof(c_short) * 5,
136        # Data format of sound
137        format=SOUND_FORMAT.PCM16.value,
138        # User callback to reading
139        pcmreadcallback=SOUND_PCMREADCALLBACK(pcmread_callback),
140        # User callback to seeking
141        pcmsetposcallback=SOUND_PCMSETPOSCALLBACK(pcmsetpos_callback),
142    )
143    sound = system.create_sound(0, mode=mode, exinfo=exinfo)
144    channel = sound.play()
145
146    subwin.clear()
147    subwin.addstr("Press SPACE to toggle pause\n" "Press q to quit")
148    row, _ = subwin.getyx()
149    while True:

```

(continues on next page)

```
150     is_playing = False
151     paused = False
152     position = 0
153     length = 0
154     if channel:
155         try:
156             is_playing = channel.is_playing
157             paused = channel.paused
158             position = channel.get_position(TIMEUNIT.MS)
159             length = sound.get_length(TIMEUNIT.MS)
160
161         except FmodError as fmoderror:
162             if not fmoderror.result is RESULT.INVALID_HANDLE:
163                 raise fmoderror
164
165     subwin.move(row + 2, 0)
166     subwin.clrtoeol()
167     subwin.addstr(
168         "Time %02d:%02d:%02d/%02d:%02d:%02d : %s"
169         % (
170             position / 1000 / 60,
171             position / 1000 % 60,
172             position / 10 % 100,
173             length / 1000 / 60,
174             length / 1000 % 60,
175             length / 10 % 100,
176             "Paused" if paused else "Playing" if is_playing else "Stopped",
177         ),
178     )
179
180     # Listen to the user
181     try:
182         keypress = subwin.getkey()
183         if keypress == " ":
184             channel.paused = not channel.paused
185         elif keypress == "q":
186             break
187     except curses.error as cerr:
188         if cerr.args[0] != "no input":
189             raise cerr
190
191     system.update()
192     time.sleep(50 / 1000)
193
194     sound.release()
195
196
197 curses.wrapper(main)
198
199 # Shut down
200 system.release()
```

INDICES AND TABLES

- genindex
- modindex
- search