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.

CHAPTER

ONE

ABOUT

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 officiale 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.

CHAPTER

TWO

INSTALLATION

To install, first make sure that you have the FMOD Engine library for you 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. :-)

CHAPTER

THREE

QUICKSTART

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

```
import pyfmodex

system = pyfmodex.System()
system.init()
sound = system.create_sound("somefile.mp3")
channel = sound.play()

while channel.is_playing:
   pass

sound.release()
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.

CHAPTER

FOUR

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.

```
"""Sample code to list identification information about all sound devices
(audio out and audio in) detected by FMOD Engine on a system.
"""

import re

import pyfmodex
from pyfmodex.enums import RESULT, SPEAKERMODE
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import DRIVER_STATE

system = pyfmodex.System()
system.init()


def _pp_driverinfo(driverinfo, indent=1):
    """Pretty print driverinfo.

    Lists all keys in the given pyfmodex.structobject with their values,
    indented by the given number times four spaces.

    .. todo:: Figure out how the GUID structure works exactly.
    """
    for key in driverinfo.keys():
        value = driverinfo[key]
        if isinstance(value, bytes):
            value = value.decode()
        elif isinstance(value, pyfmodex.structures.GUID):
            continue
        elif key == "system_rate":
            value = f"{value} kHz"
```

(continues on next page)

(continued from previous page)

```
        elif key == "speaker_mode":
            value = SPEAKERMODE(value).name
        elif key == "state":
            value = re.sub(r"^DRIVER_STATE.|\)$", "", str(DRIVER_STATE(value))).replace(
                "|", ", "
            )
        print(4 * " " * indent, end="")
        print(f"{key}: {value}")
    print()


def list_drivers(title, meth):
    """List and prettyprint information about drivers returned by the given
    method.
    """
    print(title)
    print("-" * len(title))
    counter = 0
    while True:
        try:
            driverinfo = meth(counter)
        except FmodError as fmoderr:
            if fmoderr.result == RESULT.INVALID_PARAM:
                break
            raise fmoderr
        print(f"Index {counter}:")
        _pp_driverinfo(driverinfo)
        counter += 1


list_drivers("Detected audio OUT devices", system.get_driver_info)
list_drivers("Detected audio IN devices", system.get_record_driver_info)

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.)

```
"""Sample code to show basic positioning of 3D sounds."""

import curses
import sys
import time
from math import sin

import pyfmodex
from pyfmodex.flags import MODE
```

(continues on next page)

(continued from previous page)

```

INTERFACE_UPDATETIME = 50
DISTANCEFACTOR = 1
MIN_FMOD_VERSION = 0x00020108

# Create system object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init(maxchannels=3)

THREED_SETTINGS = system.threed_settings
THREED_SETTINGS.distance_factor = DISTANCEFACTOR

# Load some sounds
sound1 = system.create_sound("media/drumloop.wav", mode=MODE.THREED)
sound1.min_distance = 0.5 * DISTANCEFACTOR
sound1.max_distance = 5000 * DISTANCEFACTOR
sound1.mode = MODE.LOOP_NORMAL

sound2 = system.create_sound("media/jaguar.wav", mode=MODE.THREED)
sound2.min_distance = 0.5 * DISTANCEFACTOR
sound2.max_distance = 5000 * DISTANCEFACTOR
sound2.mode = MODE.LOOP_NORMAL

sound3 = system.create_sound("media/swish.wav")

# Play sounds at certain positions
channel1 = system.play_sound(sound1, paused=True)
channel1.position = (-10 * DISTANCEFACTOR, 0, 0)
channel1.paused = False

channel2 = system.play_sound(sound2, paused=True)
channel2.position = (15 * DISTANCEFACTOR, 0, 0)
channel2.paused = False

# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user manipulate a simple
    environment with a listener and some sounds.
    """
    listener = system.listener(0)
    pos_ch1 = int((channel1.position[0]) / DISTANCEFACTOR) + 25
    pos_ch2 = int((channel2.position[0]) / DISTANCEFACTOR) + 25

    stdscr.clear()
```

(continues on next page)

(continued from previous page)

```
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "===========\n"
        "3D Example.\n"
        "===========\n"
        "\n"
        "Press 1 to toggle sound 1 (16bit Mono 3D)\n"
        "Press 2 to toggle sound 2 (8bit Mono 3D)\n"
        "Press 3 to play a sound (16bit Stereo 2D)\n"
        "Press h or l to move listener (when in still mode)\n"
        "Press space to toggle listener still mode\n"
        "Press q to quit"
    )

    listener_automove = True
    listener_prevposx = 0
    listener_velx = 0
    clock = 0
    while True:
        tic = time.time()

        listener_posx = listener.position[0]
        environment = list("|" + 48 * "." + "|")
        environment[pos_ch1 - 1 : pos_ch1 + 2] = list("<1>")
        environment[pos_ch2 - 1 : pos_ch2 + 2] = list("<2>")
        environment[int(listener_posx / DISTANCEFACTOR) + 25] = "L"

        stdscr.addstr(11, 0, "".join(environment))
        stdscr.addstr("\n")

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "1":
                channel1.paused = not channel1.paused
            elif keypress == "2":
                channel2.paused = not channel2.paused
            elif keypress == "3":
                system.play_sound(sound3)
            elif keypress == " ":
                listener_automove = not listener_automove
            elif keypress == "q":
                break

            if not listener_automove:
                if keypress == "h":
                    listener_posx = max(
                        -24 * DISTANCEFACTOR, listener_posx - DISTANCEFACTOR
                    )
                elif keypress == "l":
```

(continues on next page)

(continued from previous page)

```
                    listener_posx = min(
                        23 * DISTANCEFACTOR, listener_posx + DISTANCEFACTOR
                    )
            except curses.error as cerr:
                if cerr.args[0] != "no input":
                    raise cerr

            # Update the listener
            if listener_automove:
                listener_posx = sin(clock * 0.05) * 24 * DISTANCEFACTOR
            listener_velx = (listener_posx - listener_prevposx) * (
                1000 / INTERFACE_UPDATETIME
            )

            listener.position = (listener_posx, 0, 0)
            listener.velocity = (listener_velx, 0, 0)
            listener_prevposx = listener_posx

            clock += 30 * (1 / INTERFACE_UPDATETIME)
            system.update()

            toc = time.time()
            time.sleep(max(0, INTERFACE_UPDATETIME / 1000 - (toc - tic)))


curses.wrapper(main)

# Shut down
sound1.release()
sound2.release()
sound3.release()

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.)

```
"""Sample code to show how to put channels into channel groups."""

import curses
import sys
import time

import pyfmodex
from pyfmodex.flags import MODE

MIN_FMOD_VERSION = 0x00020108
```

(continues on next page)

(continued from previous 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)

```
        "======================\n"
        "\n"
        "Group A : drumloop.wav, jaguar.wav, swish.wav\n"
        "Group B : c.ogg, d.ogg, e.ogg\n"
        "\n"
        "Press a to mute/unmute group A\n"
        "Press b to mute/unmute group B\n"
        "Press m to mute/unmute master group\n"
        "Press q to quit"
    )

    while True:
        stdscr.addstr(
            12, 0, f"Channels playing: {system.channels_playing['channels']}\n"
        )

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "a":
                group_a.mute = not group_a.mute
            elif keypress == "b":
                group_b.mute = not group_b.mute
            elif keypress == "m":
                group_master.mute = not group_master.mute
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)

    # A little fade out
    if not (group_master.mute or group_a.mute and group_b.mute):
        pitch = 1.0
        volume = 1.0

        fadeout_sec = 3
        for _ in range(10 * fadeout_sec):
            group_master.pitch = pitch
            group_master.volume = volume

            volume -= 1 / (10 * fadeout_sec)
            pitch -= 0.25 / (10 * fadeout_sec)

            system.update()
            time.sleep(0.1)


curses.wrapper(main)
```

(continues on next page)

(continued from previous page)

```

# Shut down
for sound in sounds:
    sound.release()

group_a.release()
group_b.release()

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.)

```
"""Sample code to demonstrate how to set up a convolution reverb DSP and work
with it.
"""

import curses
import sys
import time
from ctypes import c_short, sizeof

import pyfmodex
from pyfmodex.enums import (CHANNELCONTROL_DSP_INDEX, DSP_CONVOLUTION_REVERB,
                            DSP_TYPE, DSPCONNECTION_TYPE, SOUND_FORMAT,
                            TIMEUNIT)
from pyfmodex.flags import MODE

MIN_FMOD_VERSION = 0x00020108

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init(maxchannels=1)

# Create a new channel group to hold the convolution DSP unit
reverbgroup = system.create_channel_group("reverb")

# Create a new channel group to hold all the channels and process the dry path
maingroup = system.create_channel_group("main")
```

(continues on next page)

(continued from previous page)

```

# Create the convolution DSP unit and set it as the tail of the channel group
reverbunit = system.create_dsp_by_type(DSP_TYPE.CONVOLUTIONREVERB)
reverbgroup.add_dsp(CHANNELCONTROL_DSP_INDEX.TAIL, reverbunit)

# Open the impulse response wav file, but use FMOD_OPENONLY as we want to read
# the data into a seperate buffer
irsound = system.create_sound("media/standrews.wav", mode=MODE.DEFAULT | MODE.OPENONLY)

# For simplicity of the example, if the impulse response is the wrong format
# just display an error
if irsound.format.format != SOUND_FORMAT.PCM16:
    print(
        "Impulse Response file is the wrong audio format. It should be 16bit"
        " integer PCM data."
    )
    sys.exit(1)

# The reverb unit expects a block of data containing a single 16 bit int
# containing the number of channels in the impulse response, followed by PCM 16
# data
short_size = sizeof(c_short)
irsound_channels = irsound.format.channels
irsound_data_length = irsound.get_length(TIMEUNIT.PCMBYTES)
irdata = (c_short * (1 + irsound_data_length))()
irsound_data = irsound.read_data(irsound_data_length)[0]

irdata[0] = irsound_channels
irdata[1:] = list(irsound_data)

reverbunit.set_parameter_data(DSP_CONVOLUTION_REVERB.PARAM_IR, irdata)

# Don't pass any dry signal from the reverb unit, instead take the dry part of
# the mix from the main signal path
reverbunit.set_parameter_float(DSP_CONVOLUTION_REVERB.PARAM_DRY, -80)

# We can now release the sound object as the reverb unit has created its
# internal data
irsound.release()

# Load up and play a sample clip recorded in an anechoic chamber
sound = system.create_sound("media/singing.wav", mode=MODE.THREED | MODE.LOOP_NORMAL)
channel = system.play_sound(sound, channel_group=maingroup, paused=True)

# Create a send connection between the channel head and the reverb unit
channel_head = channel.get_dsp(CHANNELCONTROL_DSP_INDEX.HEAD)
reverb_connection = reverbunit.add_input(channel_head, DSPCONNECTION_TYPE.SEND)

channel.paused = False

# Main loop
def main(stdscr):
```

(continues on next page)

(continued from previous page)

```
    """Draw a simple TUI, grab keypresses and let the user manipulate the
    reverb connection.
    """
    wet_volume = 1
    dry_volume = 1

    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "====================\n"
        "Convolution Example.\n"
        "====================\n"
        "\n"
        "Press k and j to change dry mix\n"
        "Press h and l to change wet mix\n"
        "Press q to quit"
    )

    while True:
        stdscr.addstr(8, 0, f"wet mix [{wet_volume:.2f}] | dry mix [{dry_volume:.2f}]")

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "h":
                wet_volume = max(wet_volume - 0.05, 0)
            elif keypress == "l":
                wet_volume = min(wet_volume + 0.05, 1)
            elif keypress == "j":
                dry_volume = max(dry_volume - 0.05, 0)
            elif keypress == "k":
                dry_volume = min(dry_volume + 0.05, 1)
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        reverb_connection.mix = wet_volume
        maingroup.volume = dry_volume

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
sound.release()
maingroup.release()
```

(continues on next page)

(continued from previous page)

```
reverbgroup.remove_dsp(reverbunit)
reverbunit.disconnect_all(inputs=True, outputs=True)
reverbunit.release()
reverbgroup.release()
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 seperately.

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.)

```
"""Example code to demonstrate how to manipulate DSP network to have two
different effects on seperately filtered, different audio paths from a single
sound.
"""

import curses
import sys
import time

import pyfmodex
from pyfmodex.enums import (CHANNELCONTROL_DSP_INDEX, DSP_MULTIBAND_EQ,
                            DSP_MULTIBAND_EQ_FILTER_TYPE, DSP_TYPE,
                            SPEAKERMODE)
from pyfmodex.flags import MODE
from pyfmodex.structobject import Structobject

MIN_FMOD_VERSION = 0x00020108

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

# In this special case we want to use stereo output and not worry about varying
# matrix sizes depending on user speaker mode
software_format = Structobject(
    sample_rate=48000, speaker_mode=SPEAKERMODE.STEREO, raw_speakers=0
)
system.software_format = software_format

```

(continues on next page)

(continued from previous page)

```
# Initialize FMOD
system.init(maxchannels=1)

sound = system.create_sound("media/drumloop.wav", mode=MODE.LOOP_NORMAL)
channel = system.play_sound(sound)

# Create the DSP effects
dsplowpass = system.create_dsp_by_type(DSP_TYPE.MULTIBAND_EQ)
dsplowpass.set_parameter_int(
    DSP_MULTIBAND_EQ.A_FILTER, DSP_MULTIBAND_EQ_FILTER_TYPE.LOWPASS_24DB
)
dsplowpass.set_parameter_float(DSP_MULTIBAND_EQ.A_FREQUENCY, 1000)
dsplowpass.set_parameter_float(DSP_MULTIBAND_EQ.A_Q, 4)

dsphighpass = system.create_dsp_by_type(DSP_TYPE.MULTIBAND_EQ)
dsphighpass.set_parameter_int(
    DSP_MULTIBAND_EQ.A_FILTER, DSP_MULTIBAND_EQ_FILTER_TYPE.HIGHPASS_24DB
)
dsphighpass.set_parameter_float(DSP_MULTIBAND_EQ.A_FREQUENCY, 4000)
dsphighpass.set_parameter_float(DSP_MULTIBAND_EQ.A_Q, 4)

# Connect up the DSP network

# When a sound is played, a subnetwork is set up in the DSP network which looks
# like this (wavetable is the drumloop sound, and it feeds its data from right
# to left):
#
# [DSPHEAD]<---[DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
group_master = system.master_channel_group
dsphead = group_master.get_dsp(CHANNELCONTROL_DSP_INDEX.HEAD)
dspchannelmixer, _ = dsphead.get_input(0)

# Now disconnect channeldsp head from the wavetable to make it look like this:
#
# [DSPHEAD]    [DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
dsphead.disconnect_from(dspchannelmixer)

# Now connect the two effects to channeldsp head and store the two connections
# this makes so we can set their matrix later

#           [DSPLOWPASS]
#          /x
# [DSPHEAD]    [DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
#          \y
#           [DSPHIGHPASS]
dsplowpassconnection = dsphead.add_input(dsplowpass)  # x
dsphighpassconnection = dsphead.add_input(dsphighpass)  # y

# Now connect the channelmixer to the 2 effects
#           [DSPLOWPASS]
#          /x          \
# [DSPHEAD]             [DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
```

(continues on next page)

(continued from previous page)

```
#           \y          /
#            [DSPHIGHPASS]

dsplowpass.add_input(dspchannelmixer)  # Ignore connection - we dont care about it.
dsphighpass.add_input(dspchannelmixer)  # Ignore connection - we dont care about it.

# Now the drumloop will be twice as loud, because it is being split into 2,
# then recombined at the end. What we really want is to only feed the
# dsphead<-dsplowpass through the left speaker for that effect, and
# dsphead<-dsphighpass to the right speaker for that effect. We can do that
# simply by setting the pan, or speaker matrix of the connections

#            [DSPLOWPASS]
#          /x=1,0      \
# [DSPHEAD]              [DSPCHANNELMIXER]<---[CHANNEL HEAD]<---[WAVETABLE - DRUMLOOP.WAV]
#          \y=0,1      /
#            [DSPHIGHPASS]

lowpassmatrix = [
    1, 0,  # output to front left: take front left input signal at 1
    0, 0,  # output to front right: silence
]
highpassmatrix = [
    0, 0,  # output to front left: silence
    0, 1,  # output to front right: take front right input signal at 1
]

# Upgrade the signal coming from the channel mixer from mono to stereo
# Otherwise the lowpass and highpass will get mono signals
dspchannelmixer.channel_format = Structobject(
    channel_mask=0, num_channels=0, source_speaker_mode=SPEAKERMODE.STEREO
)

# Now set the above matrices
dsplowpassconnection.set_mix_matrix(lowpassmatrix, 2, 2)
dsphighpassconnection.set_mix_matrix(highpassmatrix, 2, 2)

dsplowpass.bypass = True
dsphighpass.bypass = True

dsplowpass.active = True
dsphighpass.active = True

# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user manipulate the
    DSP states.
    """
    pan = 0

    stdscr.clear()
    stdscr.nodelay(True)
```

(continues on next page)

(continued from previous page)

```
    # Create small visual display
    stdscr.addstr(
        "==============================\n"
        "DSP Effect Per Speaker Sample.\n"
        "==============================\n"
        "\n"
        "Press 1 to toggle lowpass (left speaker)\n"
        "Press 2 to toggle highpass (right speaker)\n"
        "Press h and l to pan sound\n"
        "Press q to quit"
    )

    while True:
        stdscr.addstr(
            10,
            0,
            f"Lowpass (left) is {'inactive' if dsplowpass.bypass else 'active  '}",
        )
        stdscr.addstr(
            11,
            0,
            f"Highpass (right) is {'inactive' if dsphighpass.bypass else 'active  '}",
        )
        stdscr.addstr(12, 0, f"Pan is {pan:.1f} ")

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "1":
                dsplowpass.bypass = not dsplowpass.bypass
            elif keypress == "2":
                dsphighpass.bypass = not dsphighpass.bypass
            elif keypress == "h":
                pan = max(pan - 0.1, -1)
                channel.set_pan(pan)
            elif keypress == "l":
                pan = min(pan + 0.1, 1)
                channel.set_pan(pan)
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
```

(continues on next page)

(continued from previous page)

```
sound.release()
dsplowpass.release()
dsphighpass.release()
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 aply the same function to the `Channel` instead of the `ChannelGroup`.

(Adapted from sample code shipped with FMOD Engine.)

```
"""Example code to show how to apply some built in software effects to sounds.
"""

import curses
import sys
import time

import pyfmodex
from pyfmodex.enums import (DSP_MULTIBAND_EQ, DSP_MULTIBAND_EQ_FILTER_TYPE,
                            DSP_TYPE)

MIN_FMOD_VERSION = 0x00020108

# Create a System object and initialize.
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init(maxchannels=1)

mastergroup = system.master_channel_group
sound = system.create_sound("media/drumloop.wav")
channel = system.play_sound(sound)

# Create some effects to play with
dsplowpass = system.create_dsp_by_type(DSP_TYPE.MULTIBAND_EQ)
dsphighpass = system.create_dsp_by_type(DSP_TYPE.MULTIBAND_EQ)
dspecho = system.create_dsp_by_type(DSP_TYPE.ECHO)
dspflange = system.create_dsp_by_type(DSP_TYPE.FLANGE)

# Configure multiband_eq DSPs to create lowpass and highpass filters
dsplowpass.set_parameter_int(
```

(continues on next page)

(continued from previous page)

```
    DSP_MULTIBAND_EQ.A_FILTER, DSP_MULTIBAND_EQ_FILTER_TYPE.LOWPASS_24DB
)
dsplowpass.set_parameter_float(DSP_MULTIBAND_EQ.A_FREQUENCY, 1000)
dsplowpass.set_parameter_float(DSP_MULTIBAND_EQ.A_Q, 4)

dsphighpass.set_parameter_int(
    DSP_MULTIBAND_EQ.A_FILTER, DSP_MULTIBAND_EQ_FILTER_TYPE.HIGHPASS_24DB
)
dsphighpass.set_parameter_float(DSP_MULTIBAND_EQ.A_FREQUENCY, 4000)
dsphighpass.set_parameter_float(DSP_MULTIBAND_EQ.A_Q, 4)

# Add them to the master channel group.  Each time an effect is added (to
# position 0) it pushes the others down the list.
mastergroup.add_dsp(0, dsplowpass)
mastergroup.add_dsp(0, dsphighpass)
mastergroup.add_dsp(0, dspecho)
mastergroup.add_dsp(0, dspflange)

# By default, bypass all effects.  This means let the original signal go
# through without processing. It will sound 'dry' until effects are enabled by
# the user.
dsplowpass.bypass = True
dsphighpass.bypass = True
dspecho.bypass = True
dspflange.bypass = True

# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user manipulate the
    DSP states.
    """
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "================\n"
        "Effects Example.\n"
        "================\n"
        "\n"
        "Press SPACE to pause/unpause sound\n"
        "Press 1 to toggle dsplowpass effect\n"
        "Press 2 to toggle dsphighpass effect\n"
        "Press 3 to toggle dspecho effect\n"
        "Press 4 to toggle dspflange effect\n"
        "Press q to quit"
    )

    while True:
        stdscr.addstr(
            11,
            0,
```

(continues on next page)

(continued from previous page)

```
            "%-8s: lowpass[%s] highpass[%s] echo [%s] flange[%s]"
            % (
                "Paused" if channel.paused else "Playing",
                " " if dsplowpass.bypass else "x",
                " " if dsphighpass.bypass else "x",
                " " if dspecho.bypass else "x",
                " " if dspflange.bypass else "x",
            ),
        )

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == " ":
                channel.paused = not channel.paused
            elif keypress == "1":
                dsplowpass.bypass = not dsplowpass.bypass
            elif keypress == "2":
                dsphighpass.bypass = not dsphighpass.bypass
            elif keypress == "3":
                dspecho.bypass = not dspecho.bypass
            elif keypress == "4":
                dspflange.bypass = not dspflange.bypass
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
mastergroup.remove_dsp(dsplowpass)
mastergroup.remove_dsp(dsphighpass)
mastergroup.remove_dsp(dspecho)
mastergroup.remove_dsp(dspflange)

dsplowpass.release()
dsphighpass.release()
dspecho.release()
dspflange.release()

sound.release()
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.)

```
"""Example code to show how to schedule channel playback into the future with
sample accuracy. Uses several scheduled channels to synchronize two or more
sounds.
"""

import curses
import sys
import time
from enum import IntEnum

import pyfmodex
from pyfmodex.enums import TIMEUNIT
from pyfmodex.structobject import Structobject

MIN_FMOD_VERSION = 0x00020108


# pylint: disable=too-few-public-methods
class Note(IntEnum):
    """The notes we need to play our song."""

    C = 0
    D = 1
    E = 2


SONG = [
    Note.E,  # Ma-
    Note.D,  # ry
    Note.C,  # had
    Note.D,  # a
    Note.E,  # lit-
    Note.E,  # tle
    Note.E,  # lamb,
    Note.E,  # .....
    Note.D,  # lit-
    Note.D,  # tle
    Note.D,  # lamb,
    Note.D,  # .....
    Note.E,  # lit-
    Note.E,  # tle
    Note.E,  # lamb,
    Note.E,  # .....
    Note.E,  # Ma-
    Note.D,  # ry
    Note.C,  # had
```

(continues on next page)

(continued from previous page)

```
    Note.D,  # a
    Note.E,  # lit-
    Note.E,  # tle
    Note.E,  # lamb,
    Note.E,  # its
    Note.D,  # fleece
    Note.D,  # was
    Note.E,  # white
    Note.D,  # as
    Note.C,  # snow.
    Note.C,  # .....
    Note.C,  # .....
    Note.C,  # .....
]

# Create a System object and initialize.
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init(maxchannels=len(SONG))

# Get information needed later for scheduling: the mixer block size, and the
# output rate of the mixer

dsp_block_len = system.dsp_buffer_size.size
output_rate = system.software_format.sample_rate

# Load our sounds - these are just sine wave tones at different frequencies.
sounds = [None] * len(Note)
sounds[Note.C] = system.create_sound("media/c.ogg")
sounds[Note.D] = system.create_sound("media/d.ogg")
sounds[Note.E] = system.create_sound("media/e.ogg")

# Create a channelgroup that the channels will play on.  We can use this
# channelgroup as our clock reference. It also means we can pause and pitch
# bend the channelgroup, without affecting the offsets of the delays, because
# the channelgroup clock which the channels feed off, will be pausing and
# speeding up/slowing down and still keeping the children in sync.
channelgroup = system.create_channel_group("Parent")

# Play all the sounds at once! Space them apart with set delay though so that
# they sound like they play in order.
CLOCK_START = 0
for note in SONG:

    # Pick a note from our tune
```

(continues on next page)

(continued from previous page)

```
    sound = sounds[note]

    # Play the sound on the channelgroup we want to use as the parent clock
    # reference (for `delay` further down)
    channel = system.play_sound(sound, channelgroup, paused=True)

    if not CLOCK_START:
        CLOCK_START = channel.dsp_clock.parent_clock

        # Start the sound into the future, by two mixer blocks worth. Should be
        # enough to avoid the mixer catching up and hitting the clock value
        # before we've finished setting up everything. Alternatively the
        # channelgroup we're basing the clock on could be paused to stop it
        # ticking.
        CLOCK_START += dsp_block_len * 2
    else:
        # Get the length of the sound in samples
        sound_len = sound.get_length(TIMEUNIT.PCM)

        # Get the default frequency that the sound was recorded at
        freq = sound.default_frequency

        # Convert the length of the sound to 'output samples' for the output
        # timeline
        sound_len = int(sound_len / freq * output_rate)

        # Place the sound clock start time to this value after the last one
        CLOCK_START += sound_len

    # Schedule the channel to start in the future at the newly calculated
    # channelgroup clock value
    channel.delay = Structobject(dsp_start=CLOCK_START, dsp_end=0, stop_channels=False)

    # Unpause the sound. Note that you won't hear the sounds, they are
    # scheduled into the future.
    channel.paused = False

# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user manipulate the
    channel parameters.
    """
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "========================\n"
        "Gapless Playback example.\n"
        "========================\n"
        "\n"
        "Press SPACE to toggle pause\n"
```

(continues on next page)

(continued from previous page)

```
        "Press k to increase pitch\n"
        "Press j to decrease pitch\n"
        "Press q to quit"
    )

    while True:
        paused_state = "Paused" if channelgroup.paused else "Playing"

        stdscr.move(9, 0)
        stdscr.clrtoeol()
        stdscr.addstr(
            f"Channels Playing {system.channels_playing.channels} : {paused_state}"
        )

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == " ":
                # Pausing the channelgroup, as the clock parent will pause any
                # scheduled sounds from continuing. If you paused the channel,
                # this would not stop the clock it is delayed against from
                # ticking, and you'd have to recalculate the delay for the
                # channel into the future again before it was unpaused.
                channelgroup.paused = not channelgroup.paused
            elif keypress == "k":
                for _ in range(50):
                    channelgroup.pitch += 0.01
                    system.update()
                    time.sleep(10 / 1000)
            elif keypress == "j":
                for _ in range(50):
                    if channelgroup.pitch > 0.1:
                        channelgroup.pitch -= 0.01
                        system.update()
                        time.sleep(10 / 1000)
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
for sound in sounds:
    sound.release()
system.release()
```

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.)

```
"""Example code to show how to play generated tones using System.play_dsp
instead of manually connecting and disconnecting DSP units.
"""

import curses
import sys
import time

import pyfmodex
from pyfmodex.enums import DSP_OSCILLATOR, DSP_TYPE

MIN_FMOD_VERSION = 0x00020108

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init(maxchannels=1)

# Create an oscillator DSP unit for the tone
dsp = system.create_dsp_by_type(DSP_TYPE.OSCILLATOR)
dsp.set_parameter_float(DSP_OSCILLATOR.RATE, 440)  # Musical note 'A'

# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user manipulate the
    DSP parameters.
    """
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "======================\n"
        "Generate Tone Example.\n"
        "======================\n"
        "\n"
        "Press 1 to play a sine wave\n"
        "Press 2 to play a sqaure wave\n"
        "Press 3 to play a saw wave\n"
```

(continues on next page)

(continued from previous page)

```
        "Press 4 to play a triangle wave\n"
        "Press SPACE to stop the channel\n"
        "Press q to quit\n"
        "Press k and j to change volume\n"
        "Press h and l to change frequency"
    )

    channel = None
    while True:
        if channel:
            playing = "playing" if channel.is_playing else "stopped"
            volume = channel.volume
            frequency = channel.frequency
        else:
            playing = "stopped"
            volume = 0
            frequency = 0

        stdscr.move(13, 0)
        stdscr.clrtoeol()
        stdscr.addstr(f"Channel is {playing}")

        stdscr.move(14, 0)
        stdscr.clrtoeol()
        stdscr.addstr(f"Volume {volume:.2f}")

        stdscr.move(15, 0)
        stdscr.clrtoeol()
        stdscr.addstr(f"Frequency {frequency}")

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "1":
                if channel:
                    channel.stop()
                channel = system.play_dsp(dsp, paused=True)
                channel.volume = 0.5
                dsp.set_parameter_int(DSP_OSCILLATOR.TYPE, 0)
                channel.paused = False
            elif keypress == "2":
                if channel:
                    channel.stop()
                channel = system.play_dsp(dsp, paused=True)
                channel.volume = 0.125
                dsp.set_parameter_int(DSP_OSCILLATOR.TYPE, 1)
                channel.paused = False
            elif keypress == "3":
                if channel:
                    channel.stop()
                channel = system.play_dsp(dsp, paused=True)
                channel.volume = 0.125
```

(continues on next page)

(continued from previous page)

```
                dsp.set_parameter_int(DSP_OSCILLATOR.TYPE, 2)
                channel.paused = False
            elif keypress == "4":
                if channel:
                    channel.stop()
                channel = system.play_dsp(dsp, paused=True)
                channel.volume = 0.5
                dsp.set_parameter_int(DSP_OSCILLATOR.TYPE, 4)
                channel.paused = False
            elif keypress == " ":
                if channel:
                    channel.stop()
                channel = None
            elif keypress == "q":
                break

            if not channel:
                raise curses.error("no input")

            if keypress == "h":
                channel.frequency = max(channel.frequency - 500, 0)
            elif keypress == "j":
                channel.volume = max(channel.volume - 0.1, 0)
            elif keypress == "k":
                channel.volume = min(channel.volume + 0.1, 1)
            elif keypress == "l":
                channel.frequency = channel.frequency + 500
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
dsp.release()
system.release()
```

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 oppurtunity 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.)

```
"""Example code to show how to play a string of sounds together without gaps,
using `delay`, to produce a granular synthesis style trick engine effect.
"""

import curses
import random
import sys
import time

import pyfmodex
from pyfmodex.enums import RESULT, TIMEUNIT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import MODE
from pyfmodex.structobject import Structobject

MIN_FMOD_VERSION = 0x00020108

soundnames = (
    "media/granular/truck_idle_off_01.wav",
    "media/granular/truck_idle_off_02.wav",
    "media/granular/truck_idle_off_03.wav",
    "media/granular/truck_idle_off_04.wav",
    "media/granular/truck_idle_off_05.wav",
    "media/granular/truck_idle_off_06.wav",
)

# Create a System object and initialize.
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init(maxchannels=2)

output_rate = system.software_format.sample_rate
dsp_block_len = system.dsp_buffer_size.size
```

(continues on next page)

(continued from previous page)

```
master_channel_group = system.master_channel_group

sounds = []
for soundname in soundnames:
    sounds.append(system.create_sound(soundname, mode=MODE.IGNORETAGS))


def queue_next_sound(playingchannel=None):
    """Queue the next sound."""
    newsound = sounds[random.randrange(0, len(sounds))]
    newchannel = system.play_sound(newsound, paused=True)

    start_delay = 0
    if playingchannel:
        # Get the start time of the playing channel
        start_delay = playingchannel.delay.dsp_start

        # Grab the length of the playing sound, and its frequency, so we can
        # calculate where to place the new sound on the time line
        sound_len = playingchannel.current_sound.get_length(TIMEUNIT.PCM)
        freq = playingchannel.frequency

        # Now calculate the length of the sound in 'output samples'. For
        # instance,  if a 44khz sound is 22050 samples long, and the output
        # rate is 48khz, then we want to delay by 24000 output samples
        sound_len = int(sound_len / freq * output_rate)

        # Add output rate adjusted sound length to the clock value of the
        # sound that is currently playing
        start_delay += sound_len
    else:
        start_delay = newchannel.dsp_clock.parent_clock
        start_delay += 2 * dsp_block_len

    # Set the delay of the new sound to the end of the old sound
    newchannel.delay = Structobject(
        dsp_start=start_delay, dsp_end=0, stop_channels=False
    )

    # Randomize pitch/volume to make it sound more realistic / random
    newchannel.frequency *= (
        1 + random.uniform(-1, 1) * .02
    )  # @22khz, range fluctuates from 21509 to 22491

    newchannel.volume *= 1 - random.random() * 0.2  # 0.8 to 1.0

    newchannel.paused = False

    return newchannel


# Main loop
```

(continues on next page)

(continued from previous page)

```
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user manipulate the
    channel paused state.
    """
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "====================================\n"
        "Granular Synthesis SetDelay Example.\n"
        "====================================\n"
        "\n"
        "Press SPACE to toggle pause\n"
        "Press q to quit"
    )

    # Kick off the first two sounds. First one is immediate, second one will be
    # triggered to start after the first one.
    channels = []
    channels.append(queue_next_sound())
    channels.append(queue_next_sound(channels[0]))

    slot = 0
    while True:
        paused_state = "paused" if master_channel_group.paused else "playing"

        stdscr.move(7, 0)
        stdscr.clrtoeol()
        stdscr.addstr(f"Channels are {paused_state}")

        # Replace the sound that just finished with a new sound, to create
        # endless seamless stitching!
        try:
            is_playing = channels[slot].is_playing
        except FmodError as fmoderror:
            if fmoderror.result != RESULT.INVALID_HANDLE:
                raise fmoderror

        if not is_playing and not master_channel_group.paused:
            # Replace sound that just ended with a new sound, queued up to
            # trigger exactly after the other sound ends
            channels[slot] = queue_next_sound(channels[1 - slot])
            slot = 1 - slot  # flip

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == " ":
                master_channel_group.paused = not master_channel_group.paused
            elif keypress == "q":
                break
```

(continues on next page)

(continued from previous page)

```
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        # If you wait too long (longer than the length of the shortest sound),
        # you will get gaps.
        time.sleep(10 / 1000)


curses.wrapper(main)

# Shut down
for sound in sounds:
    sound.release()
system.release()
```

4.10 Load from memory

This is a sample script showing how to use the `OPENMEMORY` mode flag whe 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.)

```
"""Example code to show how to load data into memory and read it from there."""

import curses
import mmap
import sys
import time
from pathlib import Path

import pyfmodex
from pyfmodex.enums import RESULT, TIMEUNIT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import MODE
from pyfmodex.structures import CREATESOUNDEXINFO

MIN_FMOD_VERSION = 0x00020108

mediadir = Path("media")
soundnames = (
    mediadir / "drumloop.wav",
    mediadir / "jaguar.wav",
    mediadir / "swish.wav",
)

# Create a System object and initialize
```

(continues on next page)

(continued from previous page)

```
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init()

sounds = []
for filename in soundnames:
    with open(filename, mode="rb") as file_obj:
        with mmap.mmap(
            file_obj.fileno(), length=0, access=mmap.ACCESS_READ
        ) as mmap_obj:
            sounds.append(
                system.create_sound(
                    mmap_obj.read(),
                    mode=MODE.OPENMEMORY | MODE.LOOP_OFF,
                    exinfo=CREATESOUNDEXINFO(length=mmap_obj.size()),
                )
            )

# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user play the sounds."""
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "========================\n"
        "Load From Memory Example.\n"
        "========================\n"
        "\n"
        f"Press 1 to play a mono sound ({soundnames[0].stem})\n"
        f"Press 2 to play a mono sound ({soundnames[1].stem})\n"
        f"Press 3 to play a stero sound ({soundnames[2].stem})\n"
        "Press q to quit"
    )

    channel = None
    currentsound = None
    while True:
        is_playing = False
        position = 0
        length = 0
        if channel:
            try:
                is_playing = channel.is_playing
```

(continues on next page)

(continued from previous page)

```
                position = channel.get_position(TIMEUNIT.MS)
                currentsound = channel.current_sound
                if currentsound:
                    length = currentsound.get_length(TIMEUNIT.MS)

            except FmodError as fmoderror:
                if fmoderror.result not in (
                    RESULT.INVALID_HANDLE,
                    RESULT.CHANNEL_STOLEN,
                ):
                    raise fmoderror

        stdscr.move(9, 0)
        stdscr.clrtoeol()
        stdscr.addstr(
            "Time %02d:%02d:%02d/%02d:%02d:%02d : %s"
            % (
                position / 1000 / 60,
                position / 1000 % 60,
                position / 10 % 100,
                length / 1000 / 60,
                length / 1000 % 60,
                length / 10 % 100,
                "Playing" if is_playing else "Stopped",
            ),
        )
        stdscr.addstr(10, 0, f"Channel Playing {system.channels_playing.channels:-2d}")

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress in "123":
                channel = system.play_sound(sounds[int(keypress) - 1])
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
for sound in sounds:
    sound.release()
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.)

```
"""Example code to show how to play sounds on multiple speakers."""

import curses
import sys
import time

import pyfmodex
from pyfmodex.enums import RESULT, SPEAKERMODE, TIMEUNIT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import MODE

MIN_FMOD_VERSION = 0x00020108

CHOICES = (
    "Mono from front left speaker",
    "Mono from front right speaker",
    "Mono from center speaker",
    "Mono from surround left speaker",
    "Mono from surround right speaker",
    "Mono from rear left speaker",
    "Mono from rear right speaker",
    "Stereo from front speakers",
    "Stereo from front speakers (channels swapped)",
    "Stereo (right only) from center speaker",
)

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init(maxchannels=len(CHOICES))

speaker_mode = SPEAKERMODE(system.software_format.speaker_mode)

sound_mono = system.create_sound("media/drumloop.wav", mode=MODE.TWOD | MODE.LOOP_OFF)
sound_stereo = system.create_sound("media/stereo.ogg", mode=MODE.TWOD | MODE.LOOP_OFF)


def is_choice_available(choice_idx):
    """Is the given configuration choice available in the current speakermode?"""
```

(continues on next page)

(continued from previous page)

```
    if speaker_mode in (SPEAKERMODE.MONO, SPEAKERMODE.STEREO):
        return choice_idx not in (2, 3, 4, 5, 6, 9)
    if speaker_mode == SPEAKERMODE.QUAD:
        return choice_idx not in (2, 5, 6, 9)
    if speaker_mode in (SPEAKERMODE.SURROUND, SPEAKERMODE.FIVEPOINTONE):
        return choice_idx not in (5, 6)

    return True


def play_sound(choice_idx):
    """Play a sound in the given configuration choice.

    Returns the created channel.
    """
    channel = None
    if choice_idx == 0:  # Mono front left
        channel = system.play_sound(sound_mono, paused=True)
        channel.set_mix_levels_output(1, 0, 0, 0, 0, 0, 0, 0)
        channel.paused = False
    elif choice_idx == 1:  # Mono front right
        channel = system.play_sound(sound_mono, paused=True)
        channel.set_mix_levels_output(0, 1, 0, 0, 0, 0, 0, 0)
        channel.paused = False
    elif choice_idx == 2:  # Mono centre
        channel = system.play_sound(sound_mono, paused=True)
        channel.set_mix_levels_output(0, 0, 1, 0, 0, 0, 0, 0)
        channel.paused = False
    elif choice_idx == 3:  # Mono surround left
        channel = system.play_sound(sound_mono, paused=True)
        channel.set_mix_levels_output(0, 0, 0, 0, 1, 0, 0, 0)
        channel.paused = False
    elif choice_idx == 4:  # Mono surround right
        channel = system.play_sound(sound_mono, paused=True)
        channel.set_mix_levels_output(0, 0, 0, 0, 0, 1, 0, 0)
        channel.paused = False
    elif choice_idx == 5:  # Mono read left
        channel = system.play_sound(sound_mono, paused=True)
        channel.set_mix_levels_output(0, 0, 0, 0, 0, 0, 1, 0)
        channel.paused = False
    elif choice_idx == 6:  # Mono read right
        channel = system.play_sound(sound_mono, paused=True)
        channel.set_mix_levels_output(0, 0, 0, 0, 0, 0, 0, 1)
        channel.paused = False
    elif choice_idx == 7:  # Stereo format
        channel = system.play_sound(sound_stereo)
    elif choice_idx == 8:  # Stereo front channel swapped
        matrix = [0, 1,
                  1, 0]
        channel = system.play_sound(sound_stereo, paused=True)
        channel.set_mix_matrix(matrix, 2, 2)
        channel.paused = False
```

(continues on next page)

(continued from previous page)

```
    elif choice_idx == 8:  # Stereo (right only) center
        matrix = [0, 0,
                  0, 0,
                  0, 1]
        channel = system.play_sound(sound_stereo, paused=True)
        channel.set_mix_matrix(matrix, 3, 2)
        channel.paused = False
    return channel


# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user play the sounds."""
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    all_opts = speaker_mode.value >= SPEAKERMODE.SEVENPOINTONE.value
    stdscr.addstr(
        "========================\n"
        "Multiple Speaker Example.\n"
        "========================\n"
        "\n"
        "Press j or k to select mode\n"
        "Press SPACE to play the sound\n"
        "Press q to quit\n"
        "\n"
        f"Speaker mode is set to {speaker_mode.name}"
        " causing some speaker options to be unavailale"
        if not all_opts
        else ""
    )

    channel = None
    currentsound = None
    choice_idx = 0
    while True:
        stdscr.move(10, 0)
        for idx, choice in enumerate(CHOICES):
            available = is_choice_available(idx)
            sel = "-" if not available else "X" if choice_idx == idx else " "
            stdscr.addstr(f"[{sel}] {choice}\n")

        is_playing = False
        position = 0
        length = 0
        if channel:
            try:
                is_playing = channel.is_playing
                position = channel.get_position(TIMEUNIT.MS)
                currentsound = channel.current_sound
                if currentsound:
```

(continues on next page)

(continued from previous page)

```
                    length = currentsound.get_length(TIMEUNIT.MS)

            except FmodError as fmoderror:
                if fmoderror.result not in (
                    RESULT.INVALID_HANDLE,
                    RESULT.CHANNEL_STOLEN,
                ):
                    raise fmoderror

        stdscr.move(11 + len(CHOICES), 0)
        stdscr.clrtoeol()
        stdscr.addstr(
            "Time %02d:%02d:%02d/%02d:%02d:%02d : %s\n"
            % (
                position / 1000 / 60,
                position / 1000 % 60,
                position / 10 % 100,
                length / 1000 / 60,
                length / 1000 % 60,
                length / 10 % 100,
                "Playing" if is_playing else "Stopped",
            ),
        )
        stdscr.addstr(f"Channels playing: {system.channels_playing.channels:-2d}")

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "k":
                old_idx = choice_idx
                while True:
                    choice_idx = max(choice_idx - 1, 0)
                    if is_choice_available(choice_idx):
                        break
                    if choice_idx == 0:
                        choice_idx = old_idx
                        break
            elif keypress == "j":
                old_idx = choice_idx
                while True:
                    choice_idx = min(choice_idx + 1, len(CHOICES) - 1)
                    if is_choice_available(choice_idx):
                        break
                    if choice_idx == len(CHOICES) - 1:
                        choice_idx = old_idx
                        break
            elif keypress == " ":
                channel = play_sound(choice_idx)
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
```

(continues on next page)

(continued from previous page)

```
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
sound_mono.release()
sound_stereo.release()
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.)

```
"""Example code to show how to play sounds on two diferent output devices from
the same application.
"""

import curses
import sys
import time

import pyfmodex
from pyfmodex.enums import OUTPUTTYPE
from pyfmodex.flags import MODE

MIN_FMOD_VERSION = 0x00020108


def fetch_driver(stdscr, system, name=""):
    """Draw a simple TUI, grab keypresses and let the user select a driver."""
    num_drivers = system.num_drivers
    if not num_drivers:
        system.output = OUTPUTTYPE.NOSOUND
        return 0

    selected_idx = 0
    drivers = [system.get_driver_info(idx).name.decode() for idx in range(num_drivers)]
    while True:
        stdscr.addstr(4, 0, "Choose a device for system ")
        stdscr.addstr(name, curses.A_BOLD)
        stdscr.addstr(
            "\n"
```

(continues on next page)

(continued from previous page)

```
            "\n"
            "Use j and k to select\n"
            "Press SPACE to confirm\n"
            "\n"
        )

        for idx in range(num_drivers):
            sel = "X" if selected_idx == idx else " "
            stdscr.addstr(f"[{sel}] - {idx}. {drivers[idx]}\n")

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "k":
                selected_idx = max(selected_idx - 1, 0)
            elif keypress == "j":
                selected_idx = min(selected_idx + 1, num_drivers - 1)
            elif keypress == " ":
                return selected_idx
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        time.sleep(50 / 1000)


# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user play some sounds."""
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "========================\n"
        "Multiple System Example.\n"
        "========================"
    )

    # Create Sound Card A
    system_a = pyfmodex.System()
    version = system_a.version
    if version < MIN_FMOD_VERSION:
        print(
            f"FMOD lib version {version:#08x} doesn't meet "
            f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
        )
        sys.exit(1)

    system_a.driver = fetch_driver(stdscr, system_a, "System A")
    system_a.init()

```

(continues on next page)

(continued from previous page)

```
    # Create Sound Card B
    system_b = pyfmodex.System()
    system_b.driver = fetch_driver(stdscr, system_b, "System B")
    system_b.init()

    # Load one sample into each sound card
    sound_a = system_a.create_sound("media/drumloop.wav", mode=MODE.LOOP_OFF)
    sound_b = system_b.create_sound("media/jaguar.wav")

    stdscr.move(4, 0)
    stdscr.clrtobot()
    stdscr.addstr(
        "Press 1 to play a sound on device A\n"
        "Press 2 to play a sound on device B\n"
        "Press q to quit"
    )
    while True:
        stdscr.move(8, 0)
        stdscr.clrtobot()
        stdscr.addstr(
            f"Channels playing on A: {system_a.channels_playing.channels}\n"
            f"Channels playing on B: {system_b.channels_playing.channels}"
        )

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "1":
                system_a.play_sound(sound_a)
            elif keypress == "2":
                system_b.play_sound(sound_b)
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system_a.update()
        system_b.update()
        time.sleep(50 / 1000)

    # Shut down
    sound_a.release()
    system_a.release()

    sound_b.release()
    system_b.release()


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.)

```
"""Example code to show how to play streaming audio from an Internet source."""

import ctypes
import curses
import sys
import time

import pyfmodex
from pyfmodex.enums import OPENSTATE, RESULT, TAGDATATYPE, TAGTYPE, TIMEUNIT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import MODE
from pyfmodex.structobject import Structobject
from pyfmodex.structures import CREATESOUNDEXINFO

URL = "https://focus.stream.publicradio.org/focus.mp3"

MIN_FMOD_VERSION = 0x00020108

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init(maxchannels=1)

# Increase the file buffer size a little bit to account for Internet lag
system.stream_buffer_size = Structobject(size=64 * 1024, unit=TIMEUNIT.RAWBYTES)

# Increase the default file chunk size to handle seeking inside large playlist
# files that may be over 2kb.
exinfo = CREATESOUNDEXINFO(filebuffersize=1024 * 16)

tags = {}


def show_tags(stdscr, sound, channel):
    """Read and print any tags that have arrived. This could, for example,
    happen if a radio station switches to a new song.
    """
    stdscr.move(11, 0)
    stdscr.addstr("Tags:\n")
    while True:
```

(continues on next page)

(continued from previous page)

```
        try:
            tag = sound.get_tag(-1)
        except FmodError:
            break

        if tag.datatype == TAGDATATYPE.STRING.value:
            tag_data = ctypes.string_at(tag.data).decode()
            tags[tag.name.decode()] = (tag_data, tag.datalen)
            if tag.type == TAGTYPE.PLAYLIST.value and not tag.name == "FILE":
                # data point to sound owned memory, copy it before the
                # sound is released
                sound.release()
                sound = system.create_sound(
                    tag.data,
                    mode=MODE.CREATESTREAM | MODE.NONBLOCKING,
                    exinfo=exinfo,
                )
        elif tag.type == TAGTYPE.FMOD.value:
            # When a song changes, the sample rate may also change, so
            # compensate here
            if tag.name.decode() == "Sample Rate Change" and channel:
                channel.frequency = float(ctypes.string_at(tag.data).decode())

        stdscr.move(12, 0)
        stdscr.clrtobot()
        for name, value in tags.items():
            stdscr.addstr(f"{name} = '{value[0]}' ({value[1]} bytes)\n")


# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user control playback."""
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "==================\n"
        "Net Stream Example.\n"
        "==================\n"
        "\n"
        "Press SPACE to toggle pause\n"
        "Press q to quit\n"
    )

    sound = system.create_sound(
        URL, mode=MODE.CREATESTREAM | MODE.NONBLOCKING, exinfo=exinfo
    )

    channel = None
    while True:
        open_state = sound.open_state
```

(continues on next page)

(continued from previous page)

```
        is_playing = False
        position = 0
        paused = False
        if channel:
            try:
                is_playing = channel.is_playing
                position = channel.get_position(TIMEUNIT.MS)
                paused = channel.paused

                # Silence the stream until we have sufficient data for smooth
                # playback
                channel.mute = open_state.starving
            except FmodError as fmoderror:
                if fmoderror.result not in (
                    RESULT.INVALID_HANDLE,
                    RESULT.CHANNEL_STOLEN,
                ):
                    raise fmoderror
        else:
            try:
                channel = system.play_sound(sound)
            except FmodError:
                # This may fail if the stream isn't ready yet, so don't check
                # for errors
                pass

        state = ""
        if open_state.state == OPENSTATE.BUFFERING:
            state = "Buffering..."
        elif open_state.state == OPENSTATE.CONNECTING:
            state = "Connecting..."
        elif paused:
            state = "Paused"
        elif is_playing:
            state = "Playing"

        if open_state.starving:
            state += " (STARVING)"

        stdscr.move(7, 0)
        stdscr.clrtoeol()
        stdscr.addstr(
            "Time = %02d:%02d:%02d\n"
            % (
                position / 1000 / 60,
                position / 1000 % 60,
                position / 10 % 100,
            ),
        )
        stdscr.addstr(
            f"State = {state}\n"
```

(continues on next page)

(continued from previous page)

```
            f"Buffer Percentage = {open_state.percent_buffered}%"
        )

        show_tags(stdscr, sound, channel)

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == " ":
                if channel:
                    channel.paused = not channel.paused
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)

    if channel:
        channel.stop()

    stdscr.clear()
    stdscr.addstr("Waiting for sound to finish opening before trying to release it...")
    stdscr.refresh()
    while True:
        if sound.open_state.state == OPENSTATE.READY:
            break
        system.update()
        time.sleep(50 / 1000)

    sound.release()


curses.wrapper(main)

# Shut down
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.)

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

(continues on next page)

(continued from previous page)

```
"""

import curses
import sys
import time

from pathlib import Path

import pyfmodex
from pyfmodex.enums import RESULT, TIMEUNIT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import MODE

MIN_FMOD_VERSION = 0x00020108

mediadir = Path("media")
soundnames = (
    mediadir / "drumloop.wav",
    mediadir / "jaguar.wav",
    mediadir / "swish.wav",
)

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init()

sounds = []
for filename in soundnames:
    # drumloop.wav has embedded loop points which automatically turns on
    # looping so we turn it off (for all) here.
    sounds.append(system.create_sound(str(filename), mode=MODE.LOOP_OFF))


# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user play the sounds."""
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "==================\n"
        "Play Sound Example.\n"
        "==================\n"
```

(continues on next page)

(continued from previous page)

```
        "\n"
        f"Press 1 to play a mono sound ({soundnames[0].stem})\n"
        f"Press 2 to play a mono sound ({soundnames[1].stem})\n"
        f"Press 3 to play a stereo sound ({soundnames[2].stem})\n"
        "Press q to quit"
    )

    channel = None
    currentsound = None
    while True:
        is_playing = False
        position = 0
        length = 0
        if channel:
            try:
                is_playing = channel.is_playing
                position = channel.get_position(TIMEUNIT.MS)
                currentsound = channel.current_sound
                if currentsound:
                    length = currentsound.get_length(TIMEUNIT.MS)

            except FmodError as fmoderror:
                if fmoderror.result not in (
                    RESULT.INVALID_HANDLE,
                    RESULT.CHANNEL_STOLEN,
                ):
                    raise fmoderror

        stdscr.move(9, 0)
        stdscr.clrtoeol()
        stdscr.addstr(
            "Time %02d:%02d:%02d/%02d:%02d:%02d : %s"
            % (
                position / 1000 / 60,
                position / 1000 % 60,
                position / 10 % 100,
                length / 1000 / 60,
                length / 1000 % 60,
                length / 10 % 100,
                "Playing" if is_playing else "Stopped",
            ),
        )
        stdscr.addstr(10, 0, f"Channel Playing {system.channels_playing.channels:-2d}")

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress in "123":
                channel = system.play_sound(sounds[int(keypress) - 1])
            elif keypress == "q":
                break
        except curses.error as cerr:
```

(continues on next page)

(continued from previous page)

```
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
for sound in sounds:
    sound.release()
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.)

```
"""Example code to show how to simply play a stream such as an MP3 or WAV."""

import curses
import sys
import time
from pathlib import Path

import pyfmodex
from pyfmodex.enums import RESULT, TIMEUNIT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import MODE

MIN_FMOD_VERSION = 0x00020108

mediadir = Path("media")
soundnames = (
    mediadir / "drumloop.wav",
    mediadir / "jaguar.wav",
    mediadir / "swish.wav",
)

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
```

(continues on next page)

(continued from previous page)

```
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init()

# This example uses an FSB file, which is a preferred pack format for fmod
# containing multiple sounds. This could just as easily be exchanged with a
# wav/mp3/ogg file for example, but in that case you wouldn't need to check for
# subsounds. Because of the check below, this example would work with both
# types of sound file (packed vs single).
sound = system.create_stream("media/wave_vorbis.fsb", mode=MODE.LOOP_NORMAL)

sound_to_play = sound
if sound.num_subsounds:
    sound_to_play = sound.get_subsound(0)

# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user control playback."""
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "====================\n"
        "Play Stream Example.\n"
        "====================\n"
        "\n"
        "Press SPACE to toggle pause\n"
        "Press q to quit"
    )

    # Play the sound
    channel = sound_to_play.play()

    while True:
        is_playing = False
        position = 0
        length = 0
        try:
            is_playing = channel.is_playing
            position = channel.get_position(TIMEUNIT.MS)
            length = sound_to_play.get_length(TIMEUNIT.MS)
        except FmodError as fmoderror:
            if not fmoderror.result is RESULT.INVALID_HANDLE:
                raise fmoderror

        stdscr.move(7, 0)
        stdscr.clrtoeol()
        stdscr.addstr(
            "Time %02d:%02d:%02d/%02d:%02d:%02d : %s"
```

(continues on next page)

(continued from previous page)

```
            % (
                position / 1000 / 60,
                position / 1000 % 60,
                position / 10 % 100,
                length / 1000 / 60,
                length / 1000 % 60,
                length / 10 % 100,
                "Paused" if channel.paused else "Playing" if is_playing else "Stopped",
            ),
        )

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == " ":
                channel.paused = not channel.paused
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
sound_to_play.release()
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.)

```
"""Example code to show how to enumerate the available recording drivers on
this device and work with them.
"""

import curses
import sys
import time
from collections import defaultdict
```

(continues on next page)

(continued from previous page)

```
from ctypes import c_int, c_short, sizeof

import pyfmodex
from pyfmodex.enums import RESULT, SOUND_FORMAT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import DRIVER_STATE, MODE, SYSTEM_CALLBACK_TYPE
from pyfmodex.structures import CREATESOUNDEXINFO

MIN_FMOD_VERSION = 0x00020108
MAX_DRIVERS_IN_VIEW = 3
MAX_DRIVERS = 16

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init()

# Setup a callback so we can be notified if the record list has changed
def record_list_changed_callback(  # pylint: disable=unused-argument
    mysystem, callback_type, commanddata1, comanddata2, userdata
):
    """Increase a counter referenced by userdata."""
    _record_list_changed_count = c_int.from_address(userdata)
    _record_list_changed_count.value += 1

    return RESULT.OK.value


record_list_changed_count = c_int(0)
system.user_data = record_list_changed_count
system.set_callback(
    record_list_changed_callback, SYSTEM_CALLBACK_TYPE.RECORDLISTCHANGED
)


recordings = [defaultdict(bool) for _ in range(MAX_DRIVERS)]


def show_record_drivers(stdscr, selected_driver_idx, num_drivers):
    """Show an overview of detected record drivers."""
    row, _ = stdscr.getyx()
    for i in range(min(MAX_DRIVERS_IN_VIEW, num_drivers)):
        idx = (selected_driver_idx - MAX_DRIVERS_IN_VIEW // 2 + i) % num_drivers
        row += 2
        if idx == selected_driver_idx:
```

(continues on next page)

(continued from previous page)

```
            stdscr.addstr(row, 0, ">")
        driver_info = system.get_record_driver_info(idx)
        statechar = "(*) " if DRIVER_STATE(driver_info.state) & DRIVER_STATE.DEFAULT␣
↪else ""
        stdscr.addstr(row, 2, f"{idx}. {statechar}{driver_info.name.decode():41s}")
        row += 1
        stdscr.addstr(row, 2, f"{driver_info.system_rate/1000:2.1f}KHz")
        stdscr.addstr(row, 10, f"{driver_info.speaker_mode_channels}ch")
        data4 = driver_info.guid.data4.zfill(8).decode()
        stdscr.addstr(
            row,
            13,
            "{%08X-%04X-%04X-%04X-%02X%02X%02X%02X%02X%02X%02X}"
            % (
                driver_info.guid.data1,
                driver_info.guid.data2,
                driver_info.guid.data3,
                int(data4[0]) << 8 | int(data4[1]),
                int(data4[2]),
                int(data4[3]),
                int(data4[4]),
                int(data4[5]),
                int(data4[6]),
                int(data4[7]),
            ),
        )
        row += 1
        stdscr.addstr(
            row,
            2,
            "(%s) (%s) (%s)"
            % (
                "Connected"
                if (DRIVER_STATE(driver_info.state) & DRIVER_STATE.CONNECTED)
                else "Unplugged",
                "Recording" if system.is_recording(idx) else "Not recoding",
                "Playing"
                if recordings[idx]["channel"] and recordings[idx]["channel"].is_playing
                else "Not playing",
            ),
        )


# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user control recording
    and playback.
    """
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
```

(continues on next page)

(continued from previous page)

```
    stdscr.addstr(
        "============================\n"
        "Record Enumeration Example.\n"
        "============================"
    )

    selected_driver_idx = 0
    cur_y, _ = stdscr.getyx()
    while True:
        stdscr.move(cur_y + 2, 0)
        stdscr.clrtobot()
        stdscr.addstr(
            f"Record list has updated {record_list_changed_count.value} time(s)\n"
            f"Currently, {system.record_num_drivers.connected} recording device(s) are␣
↪plugged in\n"
            "\n"
            "Press j and k to scroll list\n"
            "Press q to quit\n"
            "\n"
            "Press 1 to start/stop recording\n"
            "Press 2 to start/stop playback"
        )

        # Clamp the reported number of drivers to simplify this example
        num_drivers = min(system.record_num_drivers.drivers, MAX_DRIVERS)

        subwin = stdscr.subwin(cur_y + 9, 0)
        show_record_drivers(subwin, selected_driver_idx, num_drivers)

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == "j":
                selected_driver_idx = (selected_driver_idx + 1) % num_drivers
            elif keypress == "k":
                selected_driver_idx = (selected_driver_idx - 1) % num_drivers
            elif keypress == "q":
                break
            elif keypress == "1":
                if system.is_recording(selected_driver_idx):
                    system.record_stop(selected_driver_idx)
                else:
                    # Clean up previous record sound
                    if recordings[selected_driver_idx]["sound"]:
                        recordings[selected_driver_idx]["sound"].release()

                    # Query device native settings and start a recording
                    record_driver_info = system.get_record_driver_info(
                        selected_driver_idx
                    )
                    exinfo = CREATESOUNDEXINFO(
                        numchannels=record_driver_info.speaker_mode_channels,
```

(continues on next page)

(continued from previous page)

```
                        format=SOUND_FORMAT.PCM16.value,
                        defaultfrequency=record_driver_info.system_rate,
                        # one second buffer; size here does not change the latency
                        length=record_driver_info.system_rate
                        * sizeof(c_short)
                        * record_driver_info.speaker_mode_channels,
                    )
                    sound = system.create_sound(
                        0, mode=MODE.LOOP_NORMAL | MODE.OPENUSER, exinfo=exinfo
                    )
                    recordings[selected_driver_idx]["sound"] = sound
                    try:
                        system.record_start(selected_driver_idx, sound, loop=True)
                    except FmodError as fmoderror:
                        if fmoderror.result != RESULT.RECORD_DISCONNECTED:
                            raise fmoderror
            elif keypress == "2":
                channel = recordings[selected_driver_idx]["channel"]
                sound = recordings[selected_driver_idx]["sound"]
                if channel and channel.is_playing:
                    channel.stop()
                    recordings[selected_driver_idx]["channel"] = False
                elif sound:
                    recordings[selected_driver_idx]["channel"] = sound.play()

        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)


curses.wrapper(main)

# Shut down
for recorder in recordings:
    if recorder["sound"]:
        recorder["sound"].release()
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.)

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

(continues on next page)

(continued from previous page)

```
while keeping a specified latency between the two.
"""

import curses
import sys
import time
from ctypes import c_short, sizeof

import pyfmodex
from pyfmodex.enums import RESULT, SOUND_FORMAT, TIMEUNIT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import MODE
from pyfmodex.reverb_presets import REVERB_PRESET
from pyfmodex.structures import CREATESOUNDEXINFO, REVERB_PROPERTIES

MIN_FMOD_VERSION = 0x00020108

# Some devices will require higher latency to avoid glitches
LATENCY_MS = 50
DRIFT_MS = 1
RECORD_DEVICE_INDEX = 0

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init()

if not system.record_num_drivers:
    print("No recording devices found/plugged in! Aborting.")
    sys.exit(1)

# Determine latency in samples
record_driver_info = system.get_record_driver_info(RECORD_DEVICE_INDEX)

# The point where we start compensating for drift
drift_threshold = record_driver_info.system_rate * DRIFT_MS / 1000
# User specified latency
desired_latency = record_driver_info.system_rate * LATENCY_MS / 1000

# Create user sound to record into, then start recording
exinfo = CREATESOUNDEXINFO(
    numchannels=record_driver_info.speaker_mode_channels,
    format=SOUND_FORMAT.PCM16.value,
    defaultfrequency=record_driver_info.system_rate,
    # one second buffer; size here does not change the latency
```

(continues on next page)

(continued from previous page)

```
    length=record_driver_info.system_rate
    * sizeof(c_short)
    * record_driver_info.speaker_mode_channels,
)
sound = system.create_sound(0, mode=MODE.LOOP_NORMAL | MODE.OPENUSER, exinfo=exinfo)
system.record_start(RECORD_DEVICE_INDEX, sound, loop=True)
sound_len = sound.get_length(TIMEUNIT.PCM)

# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user control playback."""
    stdscr.clear()
    stdscr.nodelay(True)

    dsp_enabled = False

    # Create small visual display
    stdscr.addstr(
        "===============\n"
        "Record Example.\n"
        "===============\n"
        "\n"
        "(Adjust LATENCY_MS in the source to compensate for stuttering)\n"
        f"(Current value is {LATENCY_MS}ms)"
    )

    reverb_on = REVERB_PROPERTIES(*REVERB_PRESET.CONCERTHALL.value)
    reverb_off = REVERB_PROPERTIES(*REVERB_PRESET.OFF.value)

    # User specified latency adjusted for driver update granularity
    adjusted_latency = desired_latency
    # Latency measured once playback begins (smoothened for jitter)
    actual_latency = desired_latency

    last_record_pos = 0
    last_play_pos = 0
    samples_recorded = 0
    samples_played = 0
    min_record_delta = sound_len
    channel = None
    while True:
        stdscr.move(7, 0)
        stdscr.clrtoeol()
        stdscr.addstr(
            f"Press SPACE to {'disable' if dsp_enabled else 'enable'} DSP effect\n"
            "Press q to quit"
        )

        # Determine how much has been recorded since we last checked
        record_pos = 0
        try:
            record_pos = system.get_record_position(RECORD_DEVICE_INDEX)
```

(continues on next page)

(continued from previous page)

```
        except FmodError as fmoderror:
            if fmoderror.result != RESULT.RECORD_DISCONNECTED:
                raise fmoderror

        record_delta = (
            record_pos - last_record_pos
            if record_pos >= last_record_pos
            else record_pos + sound_len - last_record_pos
        )
        last_record_pos = record_pos
        samples_recorded += record_delta

        if record_delta and record_delta < min_record_delta:
            # Smallest driver granularity seen so far
            min_record_delta = record_delta
            # Adjust our latency if driver granularity is high
            adjusted_latency = max(desired_latency, record_delta)

        # Delay playback until our desired latency is reached
        if not channel and samples_recorded >= adjusted_latency:
            channel = sound.play()

        if channel:
            # Stop playback if recording stops
            if not system.is_recording(RECORD_DEVICE_INDEX):
                channel.paused = True

            # Determine how much has been played since we last checked
            play_pos = channel.get_position(TIMEUNIT.PCM)
            play_delta = (
                play_pos - last_play_pos
                if play_pos >= last_play_pos
                else play_pos + sound_len - last_play_pos
            )
            last_play_pos = play_pos
            samples_played += play_delta

            # Compensate for any drift
            latency = samples_recorded - samples_played
            actual_latency = 0.97 * actual_latency + 0.03 * latency

            playbackrate = record_driver_info.system_rate
            if actual_latency < adjusted_latency - drift_threshold:
                # Play position is catching up to the record position, slow
                # playback down by 2%
                playbackrate -= playbackrate / 50
            elif actual_latency > adjusted_latency + drift_threshold:
                # Play position is falling behind the record position, speed
                # playback up by 2%
                playbackrate += playbackrate / 50
            channel.frequency = playbackrate

```

(continues on next page)

(continued from previous page)

```
        adjusted_latency_ms = int(
            adjusted_latency * 1000 / record_driver_info.system_rate
        )
        actual_latency_ms = int(actual_latency * 1000 / record_driver_info.system_rate)
        samples_recorded_s = int(samples_recorded / record_driver_info.system_rate)
        samples_played_s = int(samples_played / record_driver_info.system_rate)

        stdscr.move(10, 0)
        stdscr.clrtobot()
        stdscr.addstr(
            f"Adjusted latency: {adjusted_latency:4.0f} ({adjusted_latency_ms}ms)\n"
            f"Actual latency:   {actual_latency:4.0f} ({actual_latency_ms}ms)\n"
            "\n"
            f"Recorded: {samples_recorded:5d} ({samples_recorded_s}s)\n"
            f"Played: {samples_played:5d} ({samples_played_s}s)"
        )

        # Listen to the user
        try:
            keypress = stdscr.getkey()
            if keypress == " ":
                # Add a DSP effect -- just for fun
                dsp_enabled = not dsp_enabled
                system.set_reverb_properties(
                    0, reverb_on if dsp_enabled else reverb_off
                )
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(10 / 1000)


curses.wrapper(main)

# Shut down
sound.release()
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.)

```
"""Example code to show how to create a sound with data filled by the user."""

import curses
import sys
import time
from ctypes import c_float, c_short, sizeof
from math import sin

import pyfmodex
from pyfmodex.callback_prototypes import (SOUND_PCMREADCALLBACK,
                                          SOUND_PCMSETPOSCALLBACK)
from pyfmodex.enums import RESULT, SOUND_FORMAT, TIMEUNIT
from pyfmodex.exceptions import FmodError
from pyfmodex.flags import MODE
from pyfmodex.structures import CREATESOUNDEXINFO

MIN_FMOD_VERSION = 0x00020108

# Create a System object and initialize
system = pyfmodex.System()
VERSION = system.version
if VERSION < MIN_FMOD_VERSION:
    print(
        f"FMOD lib version {VERSION:#08x} doesn't meet "
        f"minimum requirement of version {MIN_FMOD_VERSION:#08x}"
    )
    sys.exit(1)

system.init()

# pylint: disable=invalid-name
# Using names common in mathematics
t1, t2 = c_float(0), c_float(0)  # time
v1, v2 = c_float(0), c_float(0)  # velocity


def pcmread_callback(sound_p, data_p, datalen_i):  # pylint: disable=unused-argument
    """Read callback used for user created sounds.

    Generates smooth noise.
    """

    # >>2 = 16bit stereo (4 bytes per sample)
    for _ in range(datalen_i >> 2):
        # left channel
```

(continues on next page)

(continued from previous page)

```
        stereo16bitbuffer_left = int(sin(t1.value) * 32767)
        c_short.from_address(data_p).value = stereo16bitbuffer_left
        data_p += sizeof(c_short)

        # right channel
        stereo16bitbuffer_right = int(sin(t2.value) * 32767)
        c_short.from_address(data_p).value = stereo16bitbuffer_right
        data_p += sizeof(c_short)

        t1.value += 0.01 + v1.value
        t2.value += 0.0142 + v2.value
        v1.value += sin(t1.value) * 0.002
        v2.value += sin(t2.value) * 0.002

    return RESULT.OK.value


def pcmsetpos_callback(
    sound, subsound, position, timeunit
):  # pylint: disable=unused-argument
    """Set position callback for user created sounds or to intercept FMOD's
    decoder during an API setPositon call.

    This is useful if the user calls set_position on a channel and you want to
    seek your data accordingly.
    """
    return RESULT.OK.value


# Main loop
def main(stdscr):
    """Draw a simple TUI, grab keypresses and let the user select a sound
    generation method.
    """
    stdscr.clear()
    stdscr.nodelay(True)

    # Create small visual display
    stdscr.addstr(
        "============================\n"
        "User Created Sound Example.\n"
        "============================"
    )
    stdscr.refresh()

    subwin = stdscr.derwin(4, 0)
    subwin.nodelay(True)
    subwin.addstr(
        "Sound played here is generated in realtime. It will either play as a "
        "stream which means it is continually filled as it is playing, or it "
        "will play as a static sample, which means it is filled once as the "
        "sound is created, then, when played, it will just play that short "
```

(continues on next page)

(continued from previous page)

```
        "loop of data.\n"
        "\n"
        "Press 1 to play an generated infinite stream\n"
        "Press 2 to play a static looping sample\n"
        "Press q to quit"
    )

    mode = MODE.OPENUSER | MODE.LOOP_NORMAL
    while True:
        # Listen to the user
        try:
            keypress = subwin.getkey()
            if keypress == "1":
                mode |= MODE.CREATESTREAM
                break
            if keypress == "2":
                break
            if keypress == "q":
                return
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        time.sleep(50 / 1000)

    # Create and play the sound
    numchannels = 2
    defaultfrequency = 44100
    exinfo = CREATESOUNDEXINFO(
        # Number of channels in the sound
        numchannels=numchannels,
        # Default playback rate of the sound
        defaultfrequency=defaultfrequency,
        # Chunk size of stream update in samples. This will be the amount of
        # data passed to the user callback.
        decodebuffersize=44100,
        # Length of PCM data in bytes of whole sound (for sound.get_length)
        length=defaultfrequency * numchannels * sizeof(c_short) * 5,
        # Data format of sound
        format=SOUND_FORMAT.PCM16.value,
        # User callback to reading
        pcmreadcallback=SOUND_PCMREADCALLBACK(pcmread_callback),
        # User callback to seeking
        pcmsetposcallback=SOUND_PCMSETPOSCALLBACK(pcmsetpos_callback),
    )
    sound = system.create_sound(0, mode=mode, exinfo=exinfo)
    channel = sound.play()

    subwin.clear()
    subwin.addstr("Press SPACE to toggle pause\n" "Press q to quit")
    row, _ = subwin.getyx()
    while True:
```

(continues on next page)

(continued from previous page)

```
        is_playing = False
        paused = False
        position = 0
        length = 0
        if channel:
            try:
                is_playing = channel.is_playing
                paused = channel.paused
                position = channel.get_position(TIMEUNIT.MS)
                length = sound.get_length(TIMEUNIT.MS)

            except FmodError as fmoderror:
                if not fmoderror.result is RESULT.INVALID_HANDLE:
                    raise fmoderror

        subwin.move(row + 2, 0)
        subwin.clrtoeol()
        subwin.addstr(
            "Time %02d:%02d:%02d/%02d:%02d:%02d : %s"
            % (
                position / 1000 / 60,
                position / 1000 % 60,
                position / 10 % 100,
                length / 1000 / 60,
                length / 1000 % 60,
                length / 10 % 100,
                "Paused" if paused else "Playing" if is_playing else "Stopped",
            ),
        )

        # Listen to the user
        try:
            keypress = subwin.getkey()
            if keypress == " ":
                channel.paused = not channel.paused
            elif keypress == "q":
                break
        except curses.error as cerr:
            if cerr.args[0] != "no input":
                raise cerr

        system.update()
        time.sleep(50 / 1000)

    sound.release()


curses.wrapper(main)

# Shut down
system.release()
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search