SunVOSC

Release 0.1.0

November 09, 2016

Contents

CHAPTER 1

Overview

docs	
tests	
package	

Bidirectional OSC bridge for SunVox DLL

- Free software: MIT license

1.1 Installation

```
pip install sunvosc
```

1.2 Documentation

https://sunvosc.readthedocs.io/

1.3 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

Installation

At the command line:

```
pip install sunvosc
```

CHAPTER 3

Usage

To use SunVOSC in a project:

```
import sunvosc
```

CHAPTER 4

Reference

4.1 sunvosc

CHAPTER 5

SunVOSC and SunVox DLL internals

In order to successfully work with SunVox DLL via OSC, it is important to be aware of some of the internals involved that affect the characteristics of performance.

5.1 Slots

- In each instance of SunVox DLL, there are 4 playback slots.
- All slots share the same audio output.
- Each slot has independent state for everything else, including modules, patterns, tempo, and playback state.

5.2 Internal pattern

- SunVox DLL maintains a single internal pattern that is used for playback of notes using the `sv_send_event` API call.
- It contains 16 tracks.
- Notes are played immediately upon sending an event, rather than when the next row is scheduled to play.
- You cannot send multiple events to the internal pattern that rely on another track. For example, this means if you send a note to track 0, then send an effect to track 1 using the "previous track" note command, no effect will actually occur.
- SunVOSC exposes the internal pattern for purposes of playing notes as early as possible, such as in the case of immediate performance of notes generated by a MIDI device.

5.3 Playback patterns

- SunVOSC maintains one or more *playback patterns* in slot(s) used to play events queued via OSC.
- Each has 16 tracks (or the maximum supported by the current version of SunVox).
- Each have the same number of rows, and are played in a continuous loop.
- Each is positioned at (0, 0) on the timeline.
- *Actual rows* are used to implement *virtual rows* by way of clearing and reusing rows that were already played.

- Peer initializes SunVOSC by sending a message to initialize a playback slot, specifying number of patterns and number of actual rows per pattern.
- The following sequence is repeated during playback:
  1. Peer sends message(s) to fill virtual row(s) with events.
  2. SunVOSC receives the event(s), and stores or drops each event according to these rules:
     - SunVOSC stores an event into the appropriate playback pattern if its virtual row is greater than the current playback row, and less than the current playback row plus the number of actual rows.
     - SunVOSC silently drops an event if its virtual row is less than or equal to the current playback row, or greater than the current playback row plus the number of actual rows.
  3. Upon detecting that an actual row is being played by SunVox DLL, SunVOSC will do the following:
     - Send a message to peer with the following fields:
       * virtual row number that was just played
       * maximum virtual row now available for queueing future events
     - Initialize the prior actual row with zeros, to prevent playback of past events.
- Peer is responsible for sending future events in time for them to be stored and not dropped.
- Peer is responsible for avoiding sending events that are too far in the future and thus would be dropped.

5.4 Project patterns

- If a project is loaded into a SunVOSC slot and it contains patterns that are fully or partially in the positive timeline space, they will be relocated internally to be entirely in the negative timeline space.
- For projects that already have tracks in the negative space, tracks in positive space will be relocated: the end of all relocated tracks will occur before the beginning of the existing negative tracks.
- SunVOSC maintains a relocation map, so that any request to play some or all of a row of the original timeline space can be mapped to the relocated space.
- SunVOSC calculates the maximum number of overlapping patterns in the project, and maintains that number of extra playback patterns dedicated to playing back rows from existing patterns.
- When a request to play some or all of a row of the original timeline space is received, SunVOSC will copy the tracks from each pattern in that row to playback pattern(s) at the row that immediatey succeeds that which is currently playing.

5.5 Timing

5.5.1 Tempo and ticks

- Upon slot initialization, SunVOSC will set tempo to `125` and ticks-per-line to `1`.

  This provide the highest resolution of note scheduling that directly aligns with a standard MIDI beat clock of 24 ticks per quarter note.
- Peer may set tempo and ticks-per-line at any time by queueing standard SunVox effects.
- Peer is responsible for maintaining its own knowledge of timing for the purposes of synchronizing with other systems, e.g. generating MIDI clock events.

5.5.2 LFOs

- SunVox LFOs do not automatically reset during transport commands such as start/stop.
- Peer is responsible for resetting LFO as needed via the "Set Phase" controller if LFO phase must be deterministic.

CHAPTER 6

OSC Namespace

All arguments are required unless marked as optional.

6.1 /slotN/...

Address patterns pertaining to a specific slot witin the SunVox DLL playback engine.

N must be in the range `0..3`.

6.2 Messages sent to SunVOSC from peers

6.2.1 /slotN/inform/start,si

Add an endpoint to send informational OSC messages to.

`host(s)` The hostname or IP address to inform.

`port(i)` The port to inform.

6.2.2 /slotN/inform/stop,si

Remove an endpoint so it no longer receives informational OSC messages.

`host(s)` The string-formatted hostname or IP address to stop informing.

`port(i)` The port to stop informing.

6.2.3 /slotN/init,ii[b|s]

Stops slot playback, initializes with an empty or specified project, resets playback state, resets virtual row counter to `-1`, and resets master volume to either 80 for an empty project or the master volume set by the project being loaded.

`patterns(i)` Number of playback patterns to create. Must be at least 1 if initializing an empty project, or at least the maximum number of overlapping patterns in the timeline of the loaded project.

`pattern_length(i)` Length of playback pattern(s). Must be within the range `4..4096`.

The third argument may be one of these options:

project_data(b) (optional) Content of existing SunVox project to initialize with.

project_filename(s) (optional) Filename of existing SunVox project to initialize with. Must be UTF-8 encoded.

6.2.4 /slotN/start

Starts playback.

6.2.5 /slotN/stop

Stops playback. Does not reset module state.

6.2.6 /slotN/volume,i

Sets master volume of slot.

volume(i) The volume to set.

6.2.7 /slotN/queue,iii(i|F)(i|F)(i|s|F)(i|F)(i|F)(i|F)

Queues a command for playback.

row(i) The virtual row to queue into. Must be greater than the row currently being played.

pattern(i) The pattern to queue into. Must be within the range `0..x` where *x* must be less than the number of playback patterns the slot was initialized with.

track(i) The track to queue into. Must be within the range `0..15`.

note_cmd(i|F) The note or note command to queue, or `False` if not applicable.

velocity(i|F) The velocity of the note. Must be within the range `0..128`, or `False` if not applicable.

module(i|s|F) The module to trigger. `False` if not triggering a module. If an actual module number, must be within the range `1..255`. May be a module tag instead, if actual module number is not known.

controller(i|F) The controller to set a value for. Must be within the range `1..32`, or `False` if not adjusting a controller.

effect(i|F) The note effect to apply. Must be a valid SunVox effect number, or `False` if not applying an effect. Must not be `0x30`, which stops playback; instead use the */slotN/stop* command to do so.

parameter(i|F) The 32-bit "XXYY" encoded parameter for controller or effect, or `False` if not setting a parameter. Effects that require separate "XX" and "YY" parameters must be encoded to "XXYY" form by the sender.

6.2.8 /slotN/play,i(i|F)(i|F)(i|s|F)(i|F)(i|F)(i|F)

Plays a note immediately using the SunVox DLL internal pattern.

track(i) The track to send the note to. Must be within the range `0..15`.

note_cmd(i|F) The note or note command to queue, or `False` if not applicable. Must not be an "FX to previous track" command.

velocity(i|F) The velocity of the note. Must be within the range `0..128`, or `False` if not applicable.

module(i|s|F) The module to trigger. `False` if not triggering a module. If an actual module number, must be within the range `1..255`. May be a module tag instead, if actual module number is not known.

controller(i|F) The controller to set a value for. Must be within the range `1..32`, or `False` if not adjusting a controller.

effect(i|F) The note effect to apply. Must be a valid SunVox effect number, or `False` if not applying an effect. Must not be `0x30`, which stops playback; instead use the */slotN/stop* command to do so.

parameter(i|F) The 32-bit "XXYY" encoded parameter for controller or effect, or `False` if not setting a parameter. Effects that require separate "XX" and "YY" parameters must be encoded to "XXYY" form by the sender.

6.2.9 /slotN/new_module,ss[iii]

tag(s) A UUID representing the module that will be loaded. SunVOSC will use this tag when sending a message containing the actual module number.

module_type The type of the module, exactly as it appears in SunVox. (e.g. `Generator`)

name (default: same as module_type) The name of the new module.

x (default: 512) X position of the module.

y (default: 512) Y position of the module.

z (default: 0) Z position (layer) of the module.

6.2.10 /slotN/load_module,s(b|s)[iii]

tag(s) A UUID representing the module that will be loaded. SunVOSC will use this tag when sending a message containing the actual module number.

The second argument must be **one** of the following:

synth_data(b) Content of existing SunVox project to initialize with.

synth_filename(s) Filename of existing SunVox project to initialize with. Must be UTF-8 encoded.

Optional to specify module position:

x (default: 512) X position of the module.

y (default: 512) Y position of the module.

z (default: 0) Z position (layer) of the module.

6.2.11 /slotN/connect,(i|s)(i|s)

module_from(i|s) Tag or module number of connection's source.

module_to(i|s) Tag or module number of of connection's destination.

6.2.12 /slotN/disconnect,(i|s)(i|s)

module_from(i|s) Tag or module number of connection's source.

module_to(i|s) Tag or module number of of connection's destination.

6.3 Messages sent by SunVOSC to peers

These messages are broadcast to all listeners registered to be informed.

6.3.1 /slotN/ready

(No arguments.)

Sent to indicate that the slot has been initialized.

6.3.2 /slotN/module_created,s(i|F)

`tag(s)` The tag sent when loading or creating a module.

`number(i|F)` The module number of the module that was loaded or created; or `False` if the module couldn't be loaded or created.

6.3.3 /slotN/modules_connected,ii

`module_from(i)` Tag or module number of connection's source.

`module_to(i)` Tag or module number of of connection's destination.

6.3.4 /slotN/modules_disconnected,ii

`module_from(i)` Tag or module number of connection's source.

`module_to(i)` Tag or module number of of connection's destination.

6.3.5 /slotN/started

(No arguments.)

6.3.6 /slotN/stopped

(No arguments.)

6.3.7 /slotN/played,(i|F)(i|F)

`row(i|F)` The virtual row that began playback; ‘‘False‘‘if playback hasn't started.

`frame(i|F)` The audio frame number where the row began, relative to the beginning of slot playback; ‘‘False‘‘if playback hasn't started.

This is sent once to each listener registered using */slotN/inform/start,si* immediately after SunVOSC detects that a new row is being played by SunVox DLL.

This is also sent to a new listener immediately after it's registered to be informed.

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

7.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.2 Documentation improvements

SunVOSC could always use more documentation, whether as part of the official SunVOSC docs, in docstrings, or even on the web in blog posts, articles, and such.

7.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/metrasynth/sunvosc/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

7.4 Development

To set up *sunvosc* for local development:

1. Fork sunvosc (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/sunvosc.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

7.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`) [1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

7.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 8

Authors

- Matthew Scott

CHAPTER 9

Changelog

9.1 0.1.0 (under development)

- Initial release.

CHAPTER 10

SunVOSC license

MIT License

Copyright (c) 2016 Matthew Scott and contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

S

S