Solar Flares Documentation

Release 0.1.1

Matthew Scott

Mar 06, 2018

Contents

CHAPTER 1

Overview of Solar Flares

Part of the Metrasynth project.

1.1 Purpose

Sound design and performance tools for SunVox.

1.2 Tools included

Module Polyphonist Converts most monophonic-only MetaModules into polyphonic MetaModules that support up to 16 simultaneous voices while keeping controller values synchronized.

Pattern Polyphonist Converts a pattern that uses a standard module into one that rotates through the available voices of a polyphonic MetaModule.

1.3 Tools under development

MetaModule Construction Kit A MetaModule construction kit based on a mixture of: creating modules and initial controller settings via code; direct manipulation of code inputs and controller values; customized mapping of controllers and controller groups to the final MetaModule.

FM-n Construction Kit An application of the MetaModule Construction Kit and Module Polyphonist for creating and patching n-controller polyphonic FM synthesizers.

1.4 Support for dynamic UIs

Solar Flares has no interactive UI of its own; it interacts with dynamic UIs through data structures and hints.

Metrasynth **'Solar Sails'_** is one such project: a desktop app for Linux, Mac, and Windows that wraps the Solar Flares tools.

1.5 Requirements

- Python 3.6

CHAPTER 2

MetaModule Construction Kit

(to be written)

CHAPTER 3

Particles

(Work-in-progress design document)

3.1 Introduction

Solar Flares Particles is an algorithmic music production tool. It is designed to leverage the features of the SunVox file format and DLL.

It can be used in an "online" mode for experimentation and live performance, or in an "offline" mode where a composition is rendered to WAV file(s).

It's intended to be highly tolerant of latency. This enables trusted devices to share a common Particles session. Possible uses include remote live sessions between several musicians, or using a remote audio rendering service from a lower-powered device.

Particles does not have a full-fledged performance UI. Solar Sails is an app that offers a way to use Particles.

3.2 System architecture

3.2.1 Data structures

Module map

Participants can each allocate and deallocate modules.

The module map is a mapping between `(participant, module-uuid)` pairs and actual module numbers.

Track blocks

Participants can allocate and deallocate "track blocks" of up to 16 tracks.

Particles maintains a bank of 512 tracks, via 32 patterns of 16 tracks each. It assigns blocks of them virtually to participants as needed.

Consider a scenario where track blocks are allocated in this order:

1. 7 tracks
2. 5 tracks
3. 6 tracks
4. 6 tracks
5. 2 tracks
6. 3 tracks

The tracks would be mapped to actual SunVox patterns and tracks in this way:

Pattern 1	Track 1	Block 1	Track 1
	Track 2		Track 2
	Track 3		Track 3
	Track 4		Track 4
	Track 5		Track 5
	Track 6		Track 6
	Track 7		Track 7
	Track 8	Block 2	Track 1
	Track 9		Track 2
	Track 10		Track 3
	Track 11		Track 4
	Track 12		Track 5
	Track 13	Block 5	Track 1
	Track 14		Track 2
	Track 15		(unused)
	Track 16		(unused)

Pattern 2	Track 1	Block 3	Track 1
	Track 2		Track 2
	Track 3		Track 3
	Track 4		Track 4
	Track 5		Track 5
	Track 6		Track 6
	Track 7	Block 4	Track 1
	Track 8		Track 2
	Track 9		Track 3
	Track 10		Track 4
	Track 11		Track 5
	Track 12		Track 6
	Track 13	Block 6	Track 1
	Track 14		Track 2
	Track 15		Track 3
	Track 16		(unused)

Code log

This is a log of code that was successfully executed by a participant. It's intended for storage and sharing, not for execution by other participants.

Each entry contains:

- source code
- participant ID
- creation time

Event

An event is comprised of:

- creation time
- participant ID
- participant local sequence
- row number
- event type
- event data

Event types:

- NOTE_CMD
- module create
- module delete
- module connect
- module disconnect
- row
- track block allocate
- track block release

(TODO: explain these in more detail)

NOTE_CMD events have coordinates (track block, track, row), and can be overwritten. See "row log" below for caveats.

Event log

This is an append-only log of all events generated by all participants.

Rows

This is a row of notes intended to be sent to a playback server. It is structured as a 32x16 array containing NOTE_CMDs for each track of each pattern.

During creation of a row, track blocks are mapped to actual patterns and tracks, and participant modules are mapped to actual modules.

Row log

This is an append-only log of rows to be sent to the Playback server.

Row map

This is a map of row numbers to rows. Rows can be re-rendered and overwritten, but this must occur before playback reaches a given row.

3.2.2 Processes

Participant

A participant maintains these data structures locally:

- track block map
- module map
- code log
- event log
- row log
- row map

May connect to other participants to share log entries.

May connect to a playback server to send rows, and receive feedback about played rows.

Playback server

Receives rows and events from a designated participant, and plays them using SunVox DLL.

Sends feedback about rows played to connected processes. Feedback includes accurate timing information.

Maintains an internal row map of upcoming rows to play, by directly writing them into the SunVox-managed pattern buffers.

Can play back through local audio, or provide a stream of 32-bit samples for streaming to network or disk.

3.3 Latency-tolerant performance mode

The transfer of event payloads and playback of audio are designed to be resistant to high-latency environments. Audio playback is designed to be tightly synchronized in real time between all participants.

Of course, once sound is played back for a particular row, there's no going back. The trick is to only allow performance changes to occur at some point in the future.

Each participant periodically monitors the latency between itself and its peers, and also broadcasts that to peers. Each peer chooses the maximum latency recently found, plus an additional 25%, as an event cutoff point. The peer then actively avoids creating events prior to the cutoff point.

Participants working remotely will need to work within these limitations, but will be rewarded with the pleasure of a synchronized audio experience.

CHAPTER 4

Polyphonist

CHAPTER 5

Contributing

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

5.1 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.

5.2 Documentation improvements

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

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/metrasynth/solar-flares/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 :)

5.4 Development

To set up *solar-flares* for local development:

1. Fork solar-flares (look for the “Fork” button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/solar-flares.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.

5.4.1 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`.

5.4.2 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 6

Changelog

6.1 0.1.1 (under development)

- Fix upload to PyPI.

6.2 0.1.0 (2016-11-09)

- Initial release.

CHAPTER 7

Authors

- Matthew Scott

CHAPTER 8

Indices and tables

- genindex
- modindex
- search