- MTG - Music Technology Group
- UPF - Universitat Pompeu Fabra

SOUNDSCAPE DESIGN USER MANUAL

Author: Mattia Schirosa

December 2010

1. Table of Contents

Introduction

This manual describes how to create soundscape using the MTG Soundscape Design application. The designer have to provide a score and a database.

The score provides information about soundscape geometries, subspaces and sound concepts that populate it. Each concept is described by a set of sound events (segmented samples), a sequencing structure stored in graph (see figure 1) and parameters (see section Soundscape Design parameters) that affect the sequencing structure and the synthesis engine in real-time. The set of events collected in a concept could represent several nuances of the concept.

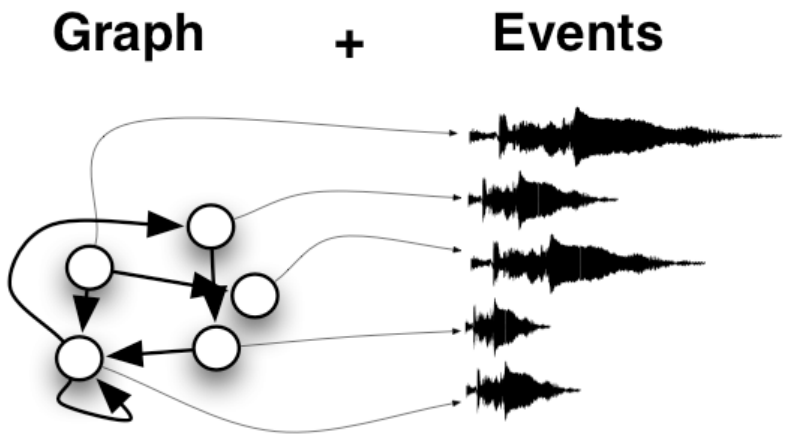

Figure 1: A Sound Concept

The database provides information about the events the user choses to simulate the concept.

The global authoring data (score + database) could be produced in two format types [1] :

- a KML plus a XML documents
- a SuperCollider patch.

Installation

The application is written in SuperCollider. We provide the application as a SuperCollider external library: *SoundscapeDesign*.

You can download it at http://mtg.upf.edu/technologies/soundscapes

Also the application uses two further SuperCollider external libraries: *GeoGraphy* and

1 The KML, namely Keyhole Markup Language, is a description open format based on XML used to display geographic data developed by Google and used on Google Earth and Google Maps.

XML. If you want to visualise graphs you need to instal SwingOSC.

You need to install:

- SuperCollider
- *SoundscapeDesign* library
- *GeoGraphy* library
- *XML* library
- (Optional) SwingOSC: http://www.sciss.de/swingOSC/ → download. (In case you don't have Java runtime environment (JRE) version 1.4 or better, read also the requirements).

To install libraries in SuperCollider you can use the Quark system (see SuperCollider Documentation) or put the library folder in the SuperCollider extension directory.

For instance put the folder *SoundscapeDesign* in the SuperCollider Extensions. The Extensions folder path is noted in the SuperCollider post window, in OSX the SuperCollider window said:

```
init_OSC
compiling class library..
    NumPrimitives = 750
    compiling dir: '/Applications/SuperCollider/SCClassLibrary'
    compiling dir: '/Library/Application Support/SuperCollider/Extensions'    ---> Extensions path
```

NOTE: to know the correct path you can evaluate the SuperCollider code:

```
Platform.userAppSupportDir
```

Soundscape Design Parameters

First, you need a name for your soundscape, and you need to defi ne its size. Second you need to define the sound space: a soundscape could be divided in zones, specifi c sub-parts that presents characteristic sound ambiances and sound sources that allows to distinguish them from the other sub-parts. They have parameters that describe the area of space they refer to (geometry, closed ambience, scale etc.). Each Zone is composed by a set of sound concepts. Concept has three types of possible position description: point source, non-point source and point source randomly generated in a area.

Finally, you need to describe the concepts sequencing.

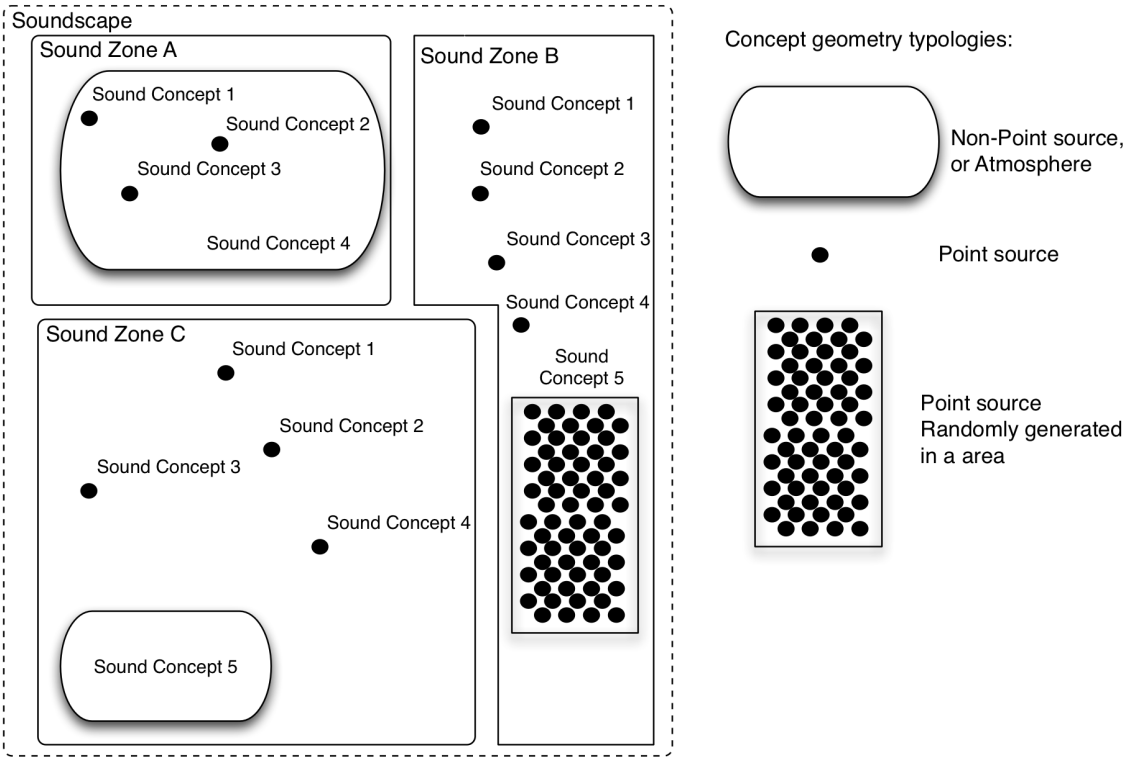

Figure 3: Sound Space

Soundscape

The following parameters must be provided, all are obligatory:

Name:

A string

The Soundscape name

Width:

A float.

Height:

A float.

Zones

for each zone the following parameters could be provided (Name is obligatory):

Name:

A string

The zone name

Geometry:

A 4d Array. a Rect -> (x,y) Upper Left corner + (Width, Height). Ex: [0,0,100,50]

or

nil - > The zone has no specific geometry, it's just a container layer for all its concepts. It allows to control all of them simultaneously. Note: nil is the SuperCollider convention for null value.

Close Ambient:

A Boolean, 1 = true.

Possibility of the zone space of being a closed ambience, 0 = open space, 1 = completely closed space. This affects the transition between zones. Mute this zone when the listener is out from its area and mute all the others zone when the listener is inside its area. The area it's gotten by the zone area.

Gain:

A Float.

0.125 = -18dB, 0.25 = -12dB, 0.5 = -6dB, 1 = 0db, 2 = +6dB, 4 = +12dB, 8 .

Parameter to amplify all the vertex of a zone, its whole loudness.

Allowed Value [0,32].

Default value is 1.

VirtualUnitMeterRatio

A Float.

The scale or the ratio between the unit of the virtual space and meters (some virtual world could use a different reference system from meters). It allows to amplify or de-amplify the attenuation relative to the distance listener-sources. Each zone could have different scale .

Ex: 0.25 means: 1 virtual unit = 0.25 meter. → Amplify

Ex: 8 means: 1 virtual unit = 8 meter. → De-amplify

Concepts

for each concept the following parameters could be provided (Name and Position are obligatory):

Name:

A string

The concept name

Gain:

A Float.

0.125 = -18dB, 0.25 = -12dB, 0.5 = -6dB, 1 = 0db, 2 = +6dB, 4 = +12dB, 8 .

Parameter to amplify all the vertex of a zone, its whole loudness.

Allowed Value [0,32]

Default value is 1.

Position:

A 2d Array or a 4d Array. Three types of position are allowed

a - Point source -> (x,y). Ex: [10,10]

b - Not-point source (also Atmosphere) -> (x,y) Upper Left Corner + (Width, Height). Ex: [10,10, 200, 40]

c - Point source Generated Random in a Area -> (x,y) Upper Left corner + (Width, Height). Ex: [10,10, 200, 40]

Point source represents a source position through a pair of coordinates. Not point source is defined by a rectangle, when the listener is inside it the concept amplitude does not change, while when the listener is outside it, the spatialisation engine computes the distance between him and the closest point of the rectangle; this position type is useful for specific sources or atmospheres that cannot be assumed to be located in a point (e.g. a waterfall). Point source random generated in an area is a useful position type to model common soundscape point sources that frequently appear in several positions moving along a specific area. For instance it is a powerful tool to model voices or waves.

Listened Area:

A Float.

The distance from which the system apply a LPF to the sound object to simulate the perception of distance.

If you want that this value be different from the default. Default value is 60 meter.

Allowed values [0, 500] meters.

Clone:

A new position Array.

If a concept is located in several fixed positions, clone is a more sensible choice than “point source random generated in an area” to model multiple concept occurrences in space. Clone allows to copy the parameters of a concept and to reuse its samples to place the same concept in another location.

Allowed Value: The position type must be the same of the father of the clone.

(WARNING: if you put clones very near one to each other combo effect could appears)

Normalisation

A float → Gain

A float → Meter

In case the is meter the value represents the distance of recording, the application compute the normalisation parameter. The parameter is used to correct the amplifi cation if the distance of recordings is

different from the default value of 5m.

Concepts Sequencing parameters

Multiple Generative Path:

A integer.

The number of concept instances playing at the same time.

Default value is 1 (this means no multiple activation).

NOTE: Multiple path should not be > the number of concept events, to avoid repetition. This is a design issue. Allowed value: [1,50]

Continuous:

A boolean.

True means Continuous. Ask the system to generate a continuous stream of events.

NOTE: Probability do not affect continuous sounds, you can just turn it on or off. Instead you could use AR, but it just adds more variation in the complex looping structure (number of in/out edges from each vertex) but doesn't provide different sequencing pauses, as the stream is continuous.

Parameter for **Non Continuous** concept:

Probability:

A float → Frequency: Times/Hours.

Allowed values [1, 3600] where the min is 1 event generation per hour and the max is 1 event generation per second.

If you don't provide the probability the application computes automatically the probability based on:

"the total duration of the this concept events" / "the total time of the database recordings".

There is also an interface with range [0,1] that is mapped to [1, 3600], with a composite function (linear from [0,0.8] and exponential from]0.8,1]). Some mapping values:

0.001	0.01	0.05	0.1	0.15	0.2	0.25	
1	2	7	13	20	26	32	
0.3	0.35	0.4	0.45	0.5	0.55	0.6	
38	44	51	57	63	69	75	
0.65	0.7	0.75	0.8	0.85	0.9	0.95	1
81	88	94	100	247	603	1473	3600

AR Arhythmic generation:

An integer.

AR controls the irregularity of the sequencing. Default AR value is 1 -> this means that the pauses between two consecutive events are all the same and the resulting sequencing is completely regular, like a metronome. If you set AR to an high value you will have both very short and very long pauses in the generation of two consecutive events. And each events will be connected with AR others.

AR in most cases should be < of the twice the number of concept instances, anyway the system raise a warning message when this value generate not meaningful sequencing pauses.

Allowed values [0,20].

Score Formats

The global authoring data (score + database) could be produced in two format types:

- a KML plus a XML documents
- a SuperCollider patch.

KML + XML Format

The application creates a soundscape starting from two annotation fi les:

1. kml score = space description and sound design parameters: zones & sound concept definition.
2. xml = database of segmented events.

The application provides a GUI for creating those annotations and to manage the database. In case you want to create the database by your own you need to:

- Create a folder named as the soundscape name
- Put the xml annotation on it (just one xml fi le could exist in this folder)
- Create all the concept folders and name them as the concept names
- Put all the events sound samples inside its relative concept folder.

Then, the SuperCollider patch you need to create to generate the soundscape is:

- create a KMLSoundscape object
- call the parse method passing the path of the kml and the path of the soundscape database:

```
(
//pass the kml annotation path
~kmlAnnotation = "/Users/mattia/virtualTravel.kml";
~soundscapeDatabase = "/Users/mattia/Canary Island Virtual Travel";
k = KMLSoundscape.new;

k.parse(~kmlAnnotation, ~soundscapeDatabase);

);
```

SuperCollider patch format

This format is specific for designer that performs soundscape analysis and recordings, but it can be also used with a database of retrieved sound. In this case skip the Recording and Studio Annotation sections.

Recording & On-site analysis

NOTE: The recordings must be Mono.

Before starts recording perform soundscape listening exploration to create the Sound Concept List. Try to discover all the events you didn't be able to imagine. Describe what you hear: takes notes, draw a score scratch, study different soundscape states, search for collection of concepts and parameters that could describes specifi c soundscape nuances.

Both Atmosphere and Events sound objects are needed. Atmospheric recordings are overall layer of sound, which cannot be analytically decomposed into single sound objects, as no particular sound signal emerges. Atmosphere characterizes quiet states without relevant sound events. Events recordings are completely focused on a specifi c source and try to avoid the presence of any background sound.

Studio Annotation

- Install *Sonic Visualiser*

NOTE: Don't use space to name concept.

In the recordings annotate segment of sound where the Sound Concept are presented:

Using Sonic Visualiser:

– Open each sound file, for all of them do:
– Create a new Region Layer, this means: in the menu LAYER -> ADD NEW REGION LAYER
– select DRAW
– draw a new region corresponding to an Instance of one of the Sound Concept you listed.
– select NAVIGATE
– double click on the segment created -> a Panel will appears
– write the Concept name in the TEXT field of the segment, you can also specify the event/instance parameters, if they are different from the parent class, the concept. In this case write the “concept name” a “coma” and all the event parameters delimitated by space . If you don't need all the fields, in SC the null value grammar is **nil**:

annotation grammar

conceptName,eventName interactionType position RecDistance rollOff

examples

wave,small splash nil 10
wave,big
wave,nil nil nil 15

– iterate for all the instances you found in the recording.

When you finished, export the annotation in .CSV and save it with the same name of the audio file it refers to.

Database Creation

There are two types of supported database format. Here we describe them and we show the relative method of the object MtgModel to use in the SuperCollider script. The first type considers the described process of studio annotation using Sonic Visualiser:

(1) Create a folder per each zone and put there all the recording and annotation files that refers to that zone.

-> relative method *MtgModel.loadAnnotationDatase(aPath)*

But you can also create a database starting from your personal sound collection or external repository. The second type of supported database is:

(2) Create a folder per each zone named as the zone, then inside this folder create a subfolder per each concept named as the concept, finally put all the relative events inside its concept folder. The events must be already segmented

-> relative method MtgModel.loadFoldersConceptDatabase(aPath)

SuperCollider script Tutorial

All the generated graph could be visualised with the object Painter of the Geography Quark. Also the graph could be created manually in case of you need, using the object GraphParser. Please, refer to the example on the Web for this subject. Here a general tutorial:

(0) Open a SuperCollider New Document, you have to execute all the code in a time, to do this write it inside parenthesis. Then, to select the code, double click just at the right of the open parenthesis, depending in your platform the execution command vary (enter on Mac, ctrl + enter on Window)

```
(

//ALL YOUR CODE

)
```

(1) Create the SoundWorld object that represents your soundscape. Specify just the arguments the Soundscape Width and the Soundscape Height and Name, like this:

```
~world = SoundWorld.new(scapeWidth:256, scapeHeight:256, name:"VirtualIsland");
```

(2) Create Environment variables (a variable that last after the 1 st code execution) for each zone of your soundscape. Also create the database environment and set the database path:

```
~zone_beach;
~zone_museum_outside;
~zone_museum_inside;

~soundscapeDatabase = "/Users/mattia/Documents/Soundscape/" ;
~beach_database = soundscapeDatabase +/+ "Beach" ;
~museum_inside_database = soundscapeDatabase +/+ "/Museum/inside";
~museum_outside_database = soundscapeDatabase +/+ "/Museum/outside" ;
```

(3) Initialize the MTG Soundscape generation system, Runner takes aGraph object as argument:

```
a = Graph.new ;
b = Runner.new(a);
```

(4) Create a MtgModel object for each zone of your soundscape. IMPORTANT: This object uses asynchronous processes (for instance when it loads up the samples in buffers) so you need to wait for them in the application side, before evaluating others commands.

You need to call all the process inside a Routine (see SuperCollider Documentation), in order to invoke waiting messages.

One way to do this is to put in a Forked function all the code that need to be executed in sequence.

```
{

//ALL FOLLOWING CODE

}.fork;
```

and wait for the sync messages required by the MtgModel asynchronous processes.

MtgModel arguments are: aRunner, name. MtgModel loadAnnotationDatase method takes the database path as argument.

The OSCresponder waiting for sync messages is no more necessary after all the zones creation, so remove it at the end.

```
//Init condition and OSCresponder to wait asynchronous threads end
c = Condition.new(false);
r  = OSCresponder(nil, '/synced',
{|time, resp, msg|
   msg.post;
   "_server osc respose".postln;
   c.test = true;
    c.signal;} ).add;


{ //forked function

//virtual_museum_outside zone
//1 asynchronous commads
~zone_museum_outside = MtgModel.new(b,name:"Museumoutside").initAudio; //a sound zone or sound layer
c.wait;
c.test = false;
//2 asynchronous commads
~zone_museum_outside.loadAnnotatedDatabase(~museum_outside_database);
c.wait;
c.test = false;
"*_* clang said: Museum outside Analysis Recordings Loaded".postln;
c.wait;
c.test = false;
```

```
	"*_* clang said: server Buffers loaded from annotation".postln;


	//virtual_museum_inside
	//1 asynchronous commads
	~zone_museum_inside = MtgModel.new(b,name:"Museuminside").initAudio; //a sound zone or
sound layer
	c.wait;
	c.test = false;
	//2 asynchronous commads
	~zone_museum_inside.loadAnnotatedDatabase(~museum_inside_database);
	c.wait;
	c.test = false;
	"*_* clang said: Museum inside Analysis Recordings Loaded".postln;
	c.wait;
	c.test = false;
	"*_* clang said: server Buffers loaded from annotation".postln;


	//beach zone
	//1 asynchronous commads
	~zone_beach = MtgModel.new(b,name:"Beach").initAudio; //a sound zone or sound layer
	c.wait;
	c.test = false;
	//2 asynchronous commads
	~zone_beach.loadAnnotatedDatabase(~beach_database);
	c.wait;
	c.test = false;
	"*_* clang said: Beach Analysis Recordings Loaded".postln;
	c.wait;
	c.test = false;
	"*_* clang said: server Buffers loaded from annotation".postln;


	r.remove; //asynchronous commands end, OSCListener remove, no more useful.
```

Note that the forked function is not closed, you have to do it at the end of your code.

(5) Add the created zones to the SoundWorld and create the Listener object.

SoundWorld.initZoneArray takes an Array of MtgModel as argument.

Listener takes the x, y coordinates of its initial position and a SoundWorld instance as arguments

```
~world.initZoneArray([~zone_beach, ~zone_museum_outside, ~zone_museum_inside]);
l = MtgListener.new(11,53, ~world);
```

(6) set the global property of your zones:

```
~zone_beach.setVirtualUnitMeterRatio(1/6);
~zone_beach.setGain(2);

~zone_museum_outside.setVirtualUnitMeterRatio(1/6);
~zone_museum_outside.setGain(2);

~zone_museum_inside.setVirtualUnitMeterRatio(1/3);
~zone_museum_inside.setGain(4);
~zone_museum_inside.setCloseAmbient([108,37,18,32]);
//this is the same as
/*
~zone_museum_inside.setGeometry([108,37,18,32]);
~zone_museum_inside.setCloseAmbient(1);
*/
```

(7) Set the sound concept parameter:

You can call the Sound Concept of a Zone using the getConcept method: it takes as argument the concept name you defined in annotations.

```
~zone_museum_outside.getConcept("SoundConceptName")
```

this will return the SoundConcept object, then to set the concept parameter use the following methods

```
setPosition([x,y]) -> arg : 2D Array = Point Source //virtual Units coordinates

setnonPointSource([x,y,W,H]) -> arg : 4D Array. Upper left corner coordinates of the
```

area & Wight and Height dimension.

setRandomPosition([x,y,W,H]) -> arg : 4D Array. Upper left corner coordinates of the area & Wight and Height dimension.

---

setMultiplePath(value) -> arg : Integer; actants

setContinuous() -> no arg : Default is False = Impulsive.

setProb(value) -> Integer; number of Times per Hour [1,3600]
setProb(value, asFreq:true) -> Float; frequency (Times/Hour) [0,1], mapped to [1,3600]

setAr() -> arg Integer; Arhythmic generation

setEventsRecordingDistance() arg Float; meter, set the same value in all concept events

setListenedArea() -> arg Integer; meter, this parameter affect the spatialisation and need to be stored as a vertex option on graph.

Clone() -> arg Position Array; the new position array, we assume the position type is the same of the father concept.

some examples :

```
//atmosphere
~zone_museum_outside.getConcept("park_atm").setnonPointSource([102,49,100,68]);
~zone_museum_outside.getConcept("park_atm").setContinuous();
~zone_museum_outside.getConcept("park_atm").setAr(8);

~zone_beach.getConcept("FlappingFlag").setPosition([114,177]);
~zone_beach.getConcept("FlappingFlag").clone([114,149]); //create new concept instance
~zone_beach.getConcept("BeachBar").setProb(50);
~zone_beach.getConcept("BeachBar").setAr(3);
~zone_beach.getConcept("BeachBar").setMultiplePath(2);
~zone_beach.getConcept("BeachBar").setListenedArea(100);
~zone_beach.getConcept("Voices").setRandomPosition([15,50,78,110]);
```

```
~zone_beach.getConcept("Voices").setEventsRecordingDistance(2);
```

(8) invoke the generation sequencing graph structures on zones, the method takes a graph instance as argument.

```
~zone_museum_inside.generateGraphs(a); //need a Graph instance
~zone_museum_outside.generateGraphs(a);
~zone_beach.generateGraphs(a);
```

(9) Start the soundscape generation: call the start method on your SoundWorld instance.

```
~world.startAllDephasingClones(); //activates the generation considering 1 sec of delay between clones their fathers in order to avoid combo effects on the same samples triggered.
```

(10) Create the soundscape Controller that allow you to control the soundscape performance. The SoundscapeController class constructor takes a SoundWolrd instance and a Boolean as arguments. The Boolean invoke the GUI.

```
~controller = SoundscapeController.new(~world, true, runner:b); //need a runner
  instance
```

The second argument ask for the GUI, if true the application creates a controller where you can interact with the gain of all the zones created. Also with the gain and probability of all the concepts created.

WARNING: Just remember to close the Forked function and the execution parenthesis, if you didn't.

```
}.fork;

)
```

(11) Now execute the code.

Performance interaction and OSC interface

The generation is active, you can start interact with the soundscape, exploring it moving the Listeners and modifying the soundscape performance.

The application provides a grammar for interacting with the created soundscape. Here the dictionary of OSC symbol, the admitted arguments and the description of the interaction functionality. The OSC interface allow to interact with the object created SoundWorld, MtgListener,and SoundscapeController.

NOTE: Replace the variable $soundscapeName$ with the name of a target soundscape.

SoundWorld messages:

```
'$soundscapeName$/add/entity/listener' → arg Int (a Listener Id)
```

Ask the SoundWorld object to add a listener to the soundscape, the listener stereophonic output channels are mapped to the Id, i.e. channel out = [id, (id+1)]. Thus the starting index for channels is ListenerId * 2.

The first Listener Id = 0 mapped to channels [0, 1], the ones are normally connected with your audio device.

```
'$soundscapeName$/remove/entity/listener' → arg Int (a Listener Id)
```

Ask the SoundWorld object to remove the listener with the provided id, the relative MtgListener object is erased and the related server synth deallocated.

MtgListener messages:

You need to replace the variable $soundscapeName$ with the name of a target soundscape and the unique listener id variable uid :

```
'$soundscapeName$/spatdif/core/listener/$uid$/position' → args two float (x,y)
```

Ask the MtgListener object to move the listener in the provided coordinates. In order to have the (0,0) starting point in the Upper Left corner fo the soundscape Rect dimension provided and due to the way SuperCollider manages GUIs coordinates (0,0 is the bottom left corner), the y coordinate goes from scapeHeight to 0 [scapeHeight, 0], so the function associated to this message compute listener y as (scapeHight – message y).

Allowed value: x [0, scapeWidth]
y [scapeHeight, 0]

Others value raise exception messages in SuperCollider post window.

`'$soundscapeName$/spatdif/core/listener/$uid$/direction'  → arg a float (angle)`

Update the listener direction, affect the spatialisation. Allowed value: [-π/2, 3π/2]

`'$soundscapeName$/spatdif/core/listener/$uid$/move' → arg a symbol`

Allowed value: [`'gostraight'`, `'goback'`, `'turnleft'`, `'turnright'`]

`'$soundscapeName$/spatdif/core/listener/$uid$/getposition' → void`

Print the position in the SuperCollider post window

SounscapeController messages:

NOTE: If you don't create the SoundscapeController object these messages have no effects. Also note that this messages are triggered by the SoundscapeController GUI, if you create it you don't need them, (step 10 of tutorial script).

You need replace the variable $soundscapeName$ with the name of a target soundscape, the variable $zoneName$ with the name of the target zone you want to control and the variable $conceptName$ with the name of the target soundscape concept you want to modify:

`'$soundscapeName$/$zoneName$/gain' → args a float (amplitude)`

Control the zone global amplitude. Allowed value: [0, 32] (See the description of the parameters above to have an explanation of the way the ranges behave)

`'$soundscapeName$/$zoneName$/$conceptName$/gain' → args a float (amplitude)`

Control the concept amplitude. Allowed value: [0, 32]

`'$soundscapeName$/$zoneName$/$conceptName$/prob' → args a float (probability)`

Control the concept probability. Allowed value: [0, 1]

Note that there is a SoundScapeController GUI that allow you to easily interact with this parameters.

Afterwards you can freeze target states of the performance (volumes and probabilities) and map them to high level presets that you can load back on demand. You need to call the savePreset and loadPreset methods on the soundscapeController object. For instance

```
~controller.savePreset("day");
~controller.savePreset("night");
~controller.savePreset("summer");
~controller.savePreset("winter");
```

```
//Afterwards
~controller.loadPreset("day");
~controller.loadPreset ("night");
~controller.loadPreset ("summer");
~controller.loadPreset ("winter");
```

Actually, we don't implement a OSC interface for the following interactions. Concept can also be moved with the SoundConcept methods:

```
~zone.getConcept(\ConceptName).move(x,y)
~zone.getConcept(\ConceptName).trasformRandomPosition(W,H)
```

Depending on the position type of the concept.

Example of OSC messages send throughout SuperCollider

```
m = NetAddr("127.0.0.1", 57120); //local host address

m.sendMsg('Canary Island Virtual Travel/add/entity/listener', 0);

m.sendMsg('Canary Island Virtual Travel/remove/entity/listener', 0);

m.sendMsg('Canary Island Virtual Travel/spatdif/core/listener/0/position',0,0);
//the upper left corner

m.sendMsg('Canary Island Virtual Travel/spatdif/core/listener/0/direction', 3.14);

m.sendMsg('Canary Island Virtual Travel/square/gain', 1);

m.sendMsg('Canary Island Virtual Travel/square/children/prob', 0.8);
```