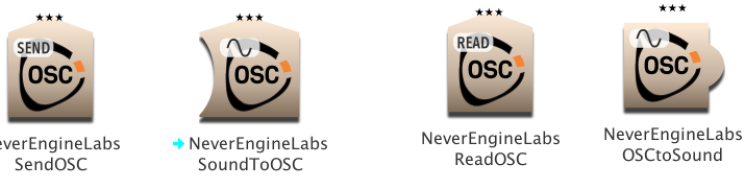
NeverEngine Labs
OSC Tools

Classes for Open Sound Control in Kyma 7

Design and Programming by Cristian Vogel and Gustav Scholda



SendOSC, ReadOSC, OSCtoSound, SoundToOSC were designed and programmed by Cristian Vogel & Gustav Scholda.
Released by NeverEngineLabs, November 2016.

Built using code from OSCHelper SmallTalk Tool for Kyma 7 (c) Symbolic Sound. Used with kind permission.
Many thanks to Carla and Kurt at SSC for their patience and encouragement.
www.neverenginelabs.com - support@neverenginlabs.com

"These tools were designed to help you compose complex OSC interactions using Kyma. I began developing them during my residency at the ZKM Institute For Music and Acoustics in February 2016. There are many compelling reasons why you would want to design OSC in Kyma. Primarily, the ability to harness the Pacarana DSP for coupling realtime software control and sonic parameters - an ideal combination for the heavy CPU demands of spatial sound composition and many other applications. Aside from the benefits of the Pacarana hardware, extending Kyma's elegant recombinant environment with OSC, is when the magic really starts to happen." - *Cristian Vogel, Copenhagen 2016*

The Kyma bidirectional OSC Protocol implemented by Symbolic Sound in 2010, seems to have been designed primarily for expressive controllers. It is straightforward and simple to use when your controller adheres to the protocol for interfacing with Kyma Sounds. To achieve a reliable connection, one of the requirements of the Kyma protocol is that the Paca(rana) will not send an OSC message until it has received that message first from a valid port and IP address. Then it knows for sure that the message is going to the right place. In most situations, this works well...but things can get complicated if you want to send many arbitrary OSC messages from Kyma to a destination that is primarily a receiver, such as a virtual world or an object based spatial audio system. That's where the NeverEngine Labs OSC Tools can help.

SendOSC and ReadOSC can send and receive arbitrary OSC messages, which you define directly in the Sound using standard OSC syntax. They will be converted transparently to the widget based syntax Kyma uses for sending and receiving OSC data.

Why use OSC syntax? The main reason is that it makes Kyma more compatible and easier to setup with established systems that have already defined a list of messages they can accept. You could copy messages from the user documentation of the system you want to control and paste them directly into the Sound parameter field to start exploring them. Working directly with OSC syntax in Kyma also makes the task of debugging connections a lot easier as you might get a better impression about the nature of messages being sent and received.

Aside from the OSC syntax parsing, our Classes also allow you to remap and rewrite OSC messages, both on the way in and the way out. You could define your own naming schemas with meaningful labelling relating aspects of how your sound design is controlling or being controlled by another complex system. Dynamic scaling and smoothing and other useful features are also available. And for more advanced programming, you can use SmallTalk to create OSC messages algorithmically using scripted variables, such as ?VoiceNumber.

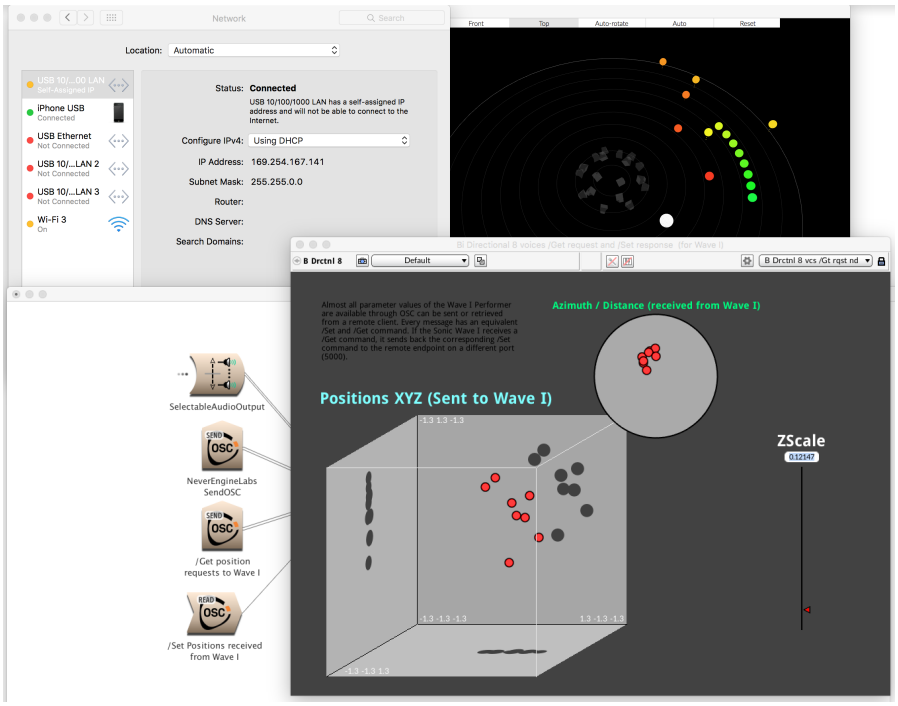
NeverEngineLabs SendOSC: Class and Parameter Description

SendOSC takes a list of destination messages in standard OSC syntax and converts the list into OSC generating widgets that Kyma can control. The Capytalk or pasted Sounds generating these widgets are sent to the OSC destination messages for controlling a receiving software or device on the same network as the Paca(rana). In the VCS, the outgoing OSC widgets will be mirrored and rewritten with your own meaningful naming schemas to be used as parameters in the rest of your Kyma sound design.

SendOSC will begin broadcasting the OSC messages that you paste into the OSCMessages field directly to the `SendToIP` and `SendToPort` you specify, without the need to have received a message from that network location first.

It does this by first parsing the OSC syntax to Kyma syntax and then generating the widgets that will transmit that translated message. You often want to use the outgoing control signals to simultaneously control parameters in the sound design. In practice the Kyma syntax can be awkward to use (for example `!osc_object_1_azimuth__1` and the like....)

SendOSC instead hides the Kyma syntax widgets and gives you the option to create copies, rewritten using a more meaningful naming schema that you define for yourself. You can then refer to the names elsewhere in your sound design, to quickly couple sound transformations in Kyma to simultaneous events in an external system. For example, a signal might control filtering on lights and sound simultaneously.



SendOSC replicated to control many 3D sources in Sonic Emotion Wave I Performer, an interface for rendering to wave field synthesis and many other VR audio formats.

Some tips for a happy sending...

- Check you have set the correct IP and Port to reach the receiver from Kyma (see parameter help for SendToIP)

- You must adhere to the OSC Syntax in the `OSCMessages` field (see parameter help for `OSCMessages`)

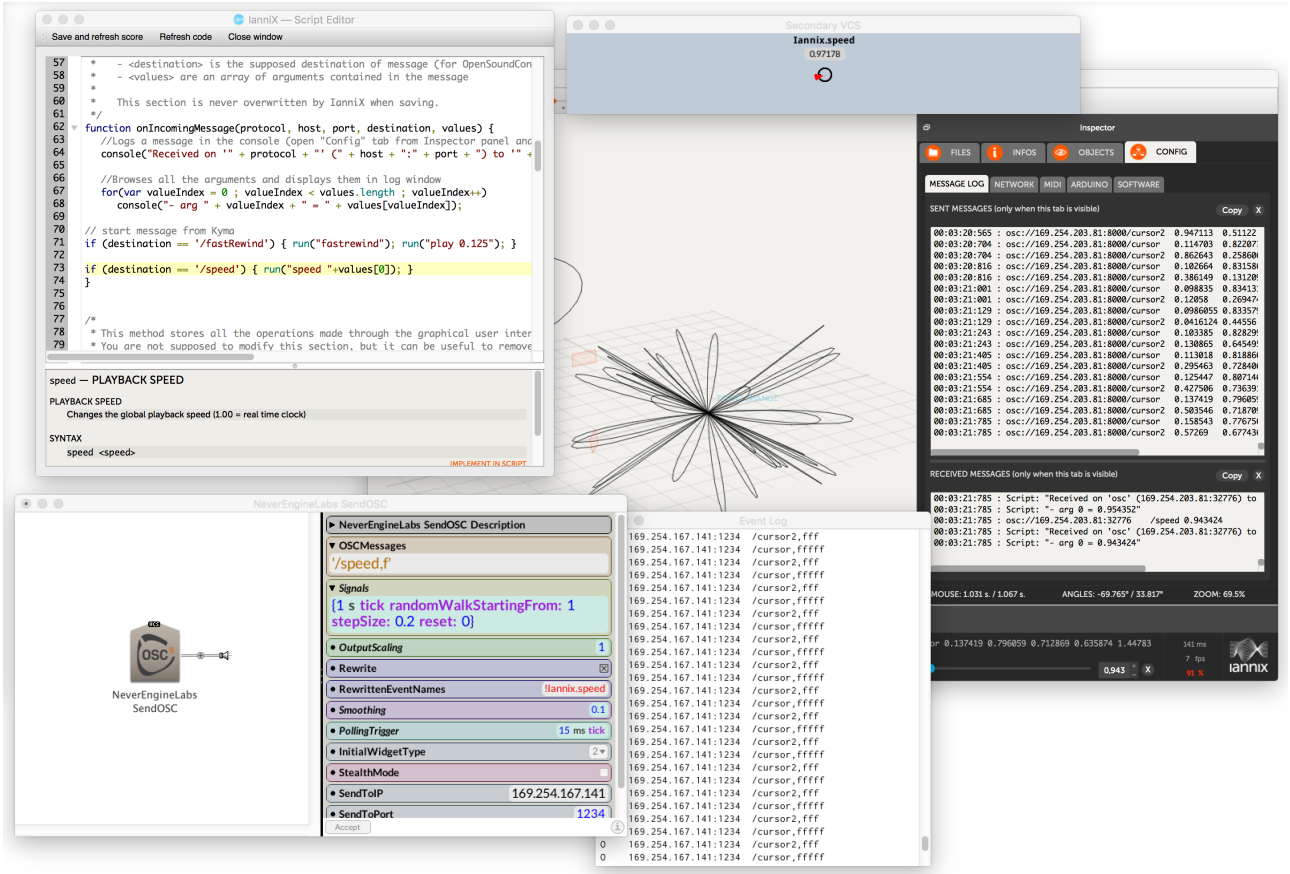
- You must have a signal associated to each OSC message AND each argument in the case that a message has more than one argument.

- If `Rewrite` is active, then you must also have a `RewrittenEventName` defined for each message **and each argument**.

- **IMPORTANT**: When choosing new names in `RewrittenEventNames` take care that those names are not used elsewhere in the signal flow, grid or timeline, or there could be an internal conflict in Kyma and may cause a crash.

- In stealth mode you can't have a completely empty VCS. You will need to have at least one widget for background OSC to keep streaming.

- A good tool for monitoring OSC network traffic is essential for debugging and programming. Kyma has Event log available from DSP Status/Configure OSC. Camille Trouillard's OSCulator is very Kyma friendly. The automation sequencer Iannix has a good built in message log too.



Kyma and Iannix are a great combination!

SendOSC Parameter Descriptions

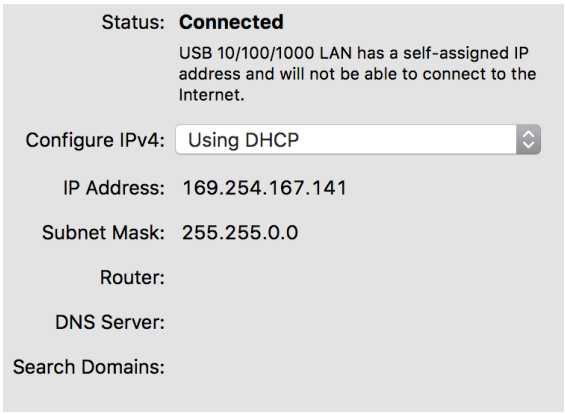
`Enable`

Enable the creation of OSC widgets and outputs. Should always be on, but can be useful to turn off the sending activity when debugging.

`SendToIP`

Here you put in the address of the device you wish to send to.

The Paca(rana) must be connected to the same network as this IP with a Cat5 ethernet cable from the ETHERNET port on the back. Make sure you haven't plugged it into the EXPANSION sockets by mistake!



TIP: This IP is not the same as the one you get from the Kyma DSP Status info.

In Mac OS, the correct IP to use is the one you will see in System Preferences/Network. The service will be an ethernet network and should say CONNECTED in bold, and will usually have a status of self-assigned IP. [1]

`SendToPort`

This is the Port number to send the signals to. Software or a device with the IP specified in the `SendToIP` parameter field, listening on this port number, will receive the Kyma signals through the `OSCMessages` you have defined.

[1] Read more about ports, IP and Kyma at http://kyma.symbolicsound.com/qa/1895/how-do-i-change-the-sending-port-for-osc

OSCMessages

Enter list of OSC messages one on each line, enclosed by single quotes.
Each message must be completed by a comma followed by the number of arguments in the message. For example, a message with three arguments:

```
'/lightingRGB,fff'
```

If the message has only one argument you still need to specify that to the OSC parser.

```
'/lamp/1/brightness,f'
```

For more advanced rewriting, you can also harness the power of replication!

Use standard SmallTalk syntax to splice green variables into your OSCMessages.
For example:

```
{ '/lamp/'&?VoiceNumber&'/brightness,f' }
```

TIP: If you are splicing in green variables, you need to put the whole expression in curly braces... and when replicating, always make sure the replicator adds a prefix to your `rewrittenEventNames (see below)`

Signals

This is the Array of CapyTalk, hotValues or pasted Sounds (let's call them signals) that will be transmitted as each OSC message.

In this field, you must enclose Capytalk expressions within curly braces, for example:
`{!RandomMod s tick nextRandom}`

Here's how it works.

The first OSC message will transmit the first signal, the second OSC message will read from from the second signal and so on. In the case that you are constructing an OSC message with two arguments, then each argument will read from a separate signal.

For example, to transmit to a virtual object in 3D space:

Three Capytalk signals describe a 3 dimensional sinusoidal panner:

```
{(1 repeatingFullRamp: 15 s) normSin}   "x signal"
{(1 repeatingFullRamp: 30 s) normCos}  "y signal"
{(1 repeatingFullRamp: 60 s) normSin}  "z signal"
```

...could transmit as one message with three arguments

```
'/virtualObjectXYZ,fff'
```

Alternatively.....

Three messages with one argument each

```
'/virtualObjectX,f'
'/virtualObjectY,f'
'/virtualObjectZ,f'
```

RewrittenEventNames

The outgoing OSC messages are transmitted by generating widgets which comply with the Kyma OSC syntax. By default, these are hidden so that you can rewrite them to more meaningfully named generated events, and refer to those names as sound parameters throughout your signal flow.

TIPS:

- You must have the same amount of generated events as there are signals going out. This will be defined by the message list you make at the `OSCMessages` parameter field (see the `OSCMessages` help description)

- You cannot perform any Capytalk transformations here. Enter `!HotValues` with exclamation marks and separated by a new line. The syntax colouring will be red, if you have done it correctly. You can add notes here in double quotes to remind you what things are.

- Make double sure you have not used a name here that already exists as a generated event somewhere else in the Sound, as you may get a conflict and possible crash.

- **Do not refer to one of your `ReWrittenEventNames` in one of the other fields of the same instance of SendOSC which is declaring it. Pacarana might get locked in an infinite loop.**

Rewrite

When `Rewrite` is active, duplicates of the Kyma syntax OSC widgets are created using the GeneratedEvent names you define in the `ReWrittenEventNames` field.

If you need to, you can disable the creation of rewritten VCS widgets, and work with the Kyma OSC widgets instead. The list in `rewrittenEventNames` is ignored in this case and you can clear that field.

- When `Rewrite` is active the Kyma syntax OSC widgets are hidden from VCS.

- When in `StealthMode`, **ALL** OSC linked widgets are hidden from the VCS.

`SignalMinValues SignalMaxValues`
`OutputMinValues OutputMaxValues`

Use the input output matrix to remap the values generated from the Kyma signals, so they extend into a meaningful range in the receiving system.

TIP: If there are not enough remapping values to match each generated argument and message, then the last value in the array will be repeated as necessary. Use this feature as a shortcut when needed.

These fields only accept arrays of fixed constants. They will not accept dynamic hotValues. But you could use Smalltalk to construct the Array algorithmically. For example:

```
{ 1 to: 10 collect: [:i |1.0 / i] }
```

would represent a linear scaling over 10 values

```
(1.0 0.5 0.333333 0.25 0.2 0.166667 0.142857 0.125 0.111111 0.1)
```

`Another` example:

In Sonic Emotions Wave I Performer an WFS object can be placed using Azimuth (degrees), Distance (centre to full distance in metres) and Elevation (ground to height in metres).

```
{ (1 repeatingFullRamp: !spinRate s) normSin }  --> (-1,1) --> (-360,360)
{ (1 repeatingFullRamp: !throwRate s) normCos }  --> (-1,1) --> (0,8)
{ (1 repeatingFullRamp: !heightRate s) normSin }  --> (-1,1) --> (0,90)
```

This matrix would quickly create some nicely mapped 3D spiralling trajectories over the speaker array.

Scale

A useful array that allows the dynamic scaling of each Kyma signal *before* it gets remapped through the input output matrix and transmitted as the generated OSC message.

You can use it as a way of fading outgoing OSC signals in or modulating them with an LFO or with pulses, whilst adhering to the limits of the remapping matrix.

TIP: If there are less values in the OutputScalingArray than there are messages, the output scale of each extra message will be filled with a copy of the last value in the array.

For example, you could create a mute for the first message (lets say a message to dim the lights in the theatre) but leave all the rest of the control messages unscaled by adding only

{`!DimLights smooth: 3 s`} followed by a 1 to fill the rest of the array.

Or create a dynamic random jitter on all scales by adding only one Capytalk expression such as `{0.1 s random seed: 0.2 smoothed}`

Alternatively you could also scale your signals in the Signals array before they get remapped using Capytalk.

Smoothing

Convenient smooth on all the outgoing signals with a duration in seconds.
Set to 0 to bypass smoothing.

PollingTrigger

Messages are only sent when this trigger changes from 0 to positive. Use it to reduce data over the network if necessary... `25 ms tick` is usually a pretty reliable polling rate. Alternatively use it as a way of muting or pausing the outputting of all messages specified in this instance of SendOSC. For example: `25 ms tick * !mute`

StealthMode

When you have finished routing and are happy with the OSC configuration, go into stealth mode to hide all OSC controlling generated widgets from the VCS.

They will continue to broadcast their values to the `RewrittenEventNames` you have specified AND transparently to the OSC you have specified. Even though they are hidden, you can still refer elsewhere in your sound design to the signals being sent out over OSC by using the more meaningful names that you defined in `RewrittenEventNames` .

For example, map something like `!VirtualObjectSize` to an object in a virtual world, a filter cut off and a reverb decay at the same time, whilst keeping your VCS clutter-free.

TIP: In stealth mode you can't have a completely empty VCS. You will need to have at least one widget for background OSC to keep streaming.

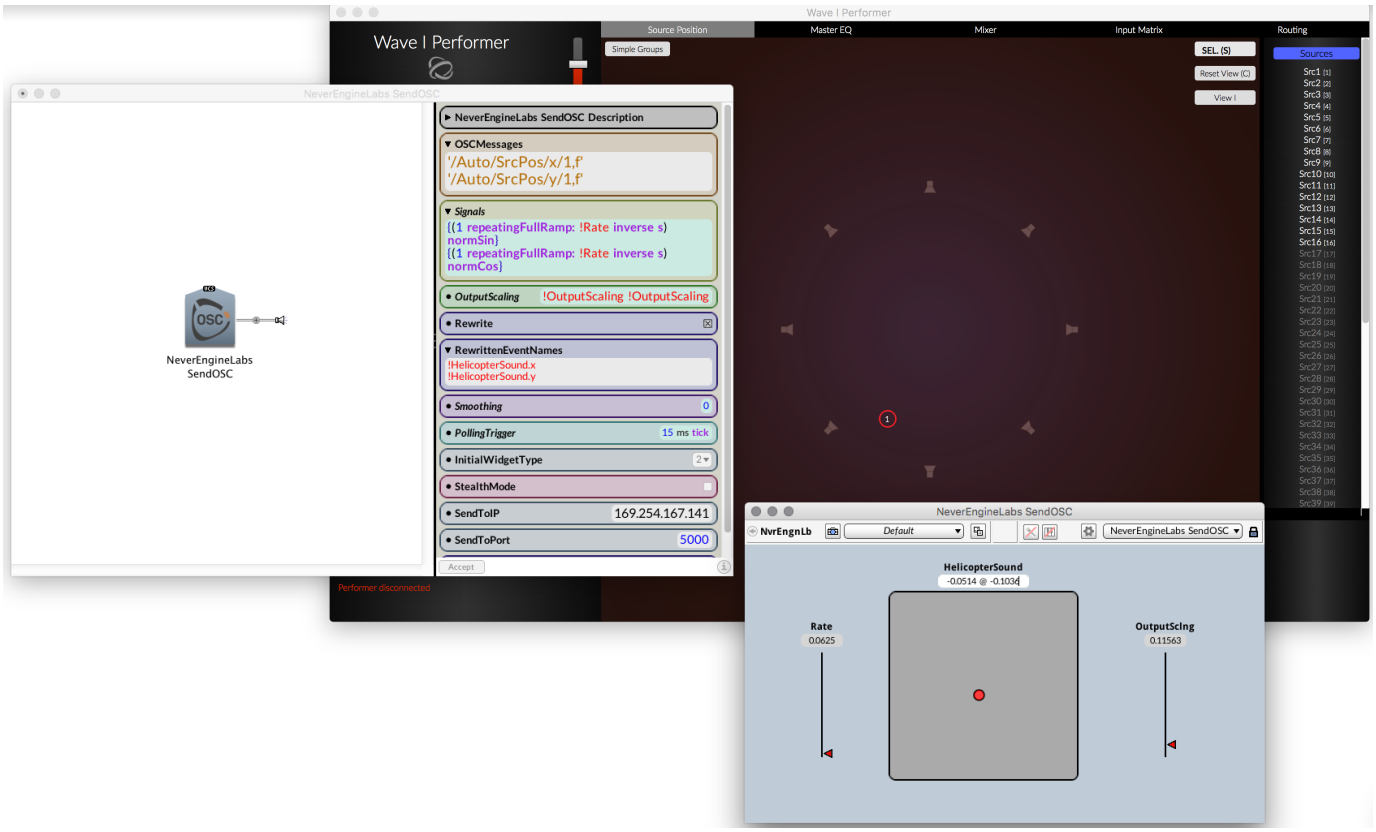
InitialWidgetType

This option lets you decide what type of Widgets will be created the first time the Sound is run. Its handy so as you don't fill up the VCS with many widgets.

Each widget will remember its display type after the first time it is created, so you will have to change the type by hand in the VCS after that.

1 - SmallFader (only numbers)
2 - Panpot
3 - Gate
4 - Fader

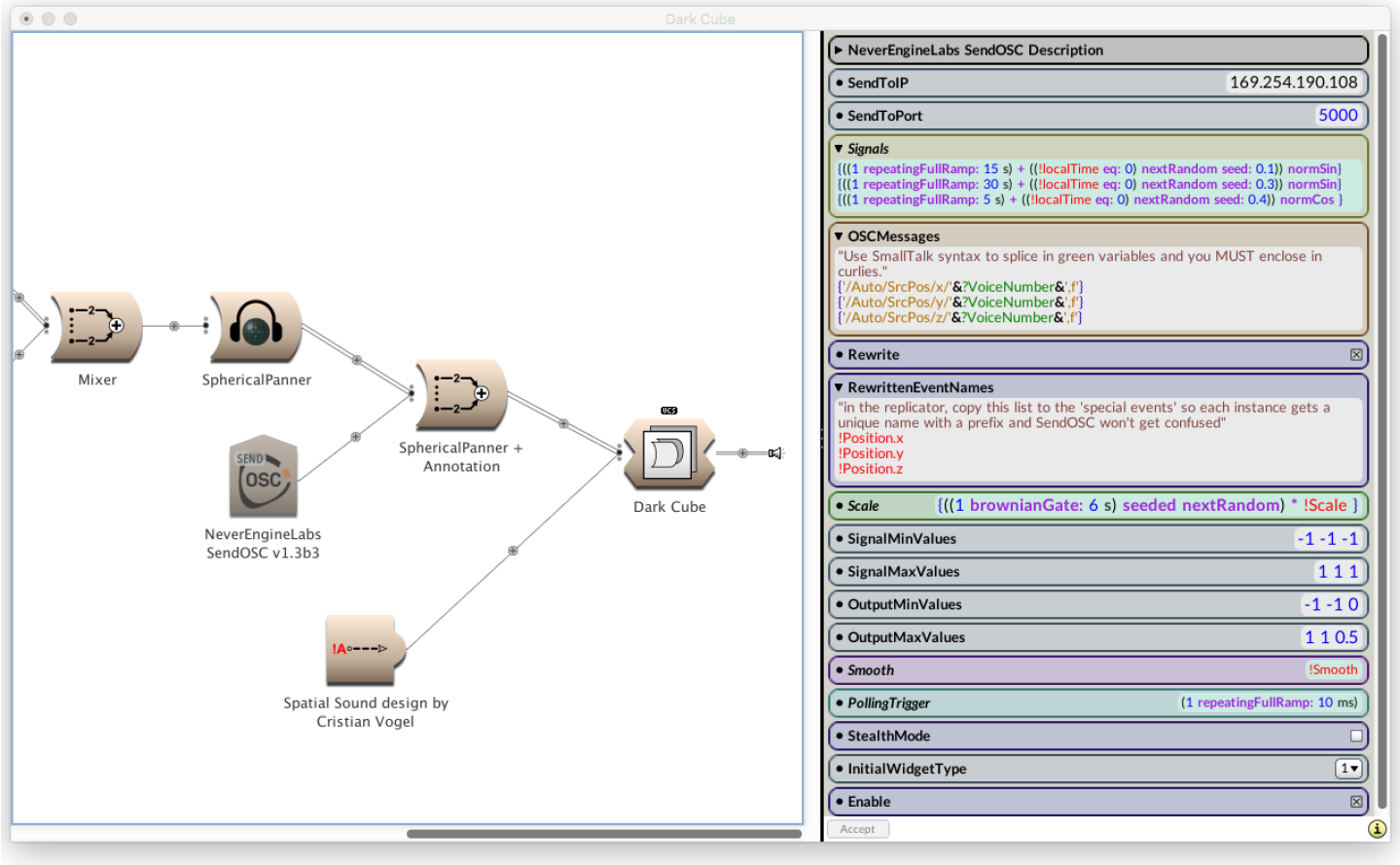
Example 1: Rewriting



Original OSC message:
'`/Set/SrcPos/1/xy,ff`'
A message accepted by Sonic Emotions Wave I Performer for setting the position of a 2D point source. The comma ff means this message has two arguments

Kyma syntax generated:
'`!osc_set_srcpos_1_xy__2`' and '`!osc_set_srcpos_1_xy__2`'
created by SendOSC (hidden if `Rewrite` *is active)*

Your names:
'`!helicopterSound.x`' and '`!helicopterSound.y`'
in Kyma 7.1 a `.x` *and* `.y` *tag will automatically create a 2D widget*



Example 2: Dark Cube

Replicating 8 instances of a moody sound design using reverbs and shortwave static reading from eight spherical sinusoids of different rates and offsets. The scale of each dimension is randomly modulated roughly every 6 seconds. The mapping matrix constrains the Z dimension to a more narrow range than the X and Y dimension.

This composition example spatialises audio in realtime using the excellent Kyma spherical panner binaural/quad renderer whilst simultaneously transmitting OSC for Sonic Emotions Wave I. This is essentially one way to compose spatial sound design on headphones in the comfort of your own studio, for later rendering in realtime over a high speaker count wave field synthesis installation.

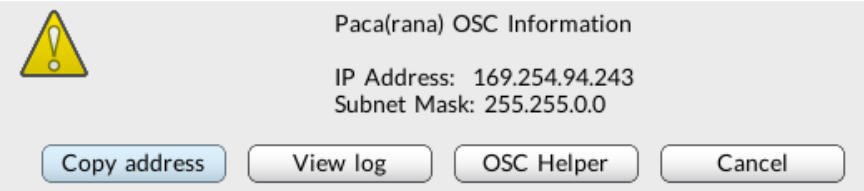
NeverEngineLabs ReadOSC: Class and Parameter Description

ReadOSC takes a list of messages in standard OSC syntax and converts them into Kyma GeneratedEvents, with value remapping, rewriting, smoothing and scaling options.

Some tips for happy reading:

- Make sure your Paca(rana) and the sending device are connected to the same network. Use the Paca(rana)'s ethernet port on the back and connect it to either a router or directly to the device (most likely your computer)

- Make sure the sending software or device is transmitting its OSC data using port **8000** (Kyma only listens on 8000) to your Paca(rana)'s IP. Kyma will tell you the IP address of your Paca(rana).



At the start of a session, first thing to do is get your Paca(rana) OSC Information from DSP Status/Configure/OSC. Write it down or paste it into the software that is sending!

TIP: The Paca(rana) has a new IP every time it is switched on.

- Adhere to the OSC Syntax in the OSCMessage field (see parameter help for OSCMessage)

- When choosing names for the OSC controlled GeneratedEvents, you must be **absolutely certain** that those names are **not already existing as generated events** elsewhere in your signal flow, multigrid and timeline - or you risk a conflict that could crash Kyma.

OSCMessages

Enter list of OSC messages you would like to read, one on each line, enclosed by single quotes.

Each message must be completed by a comma followed by the number of arguments in the message.

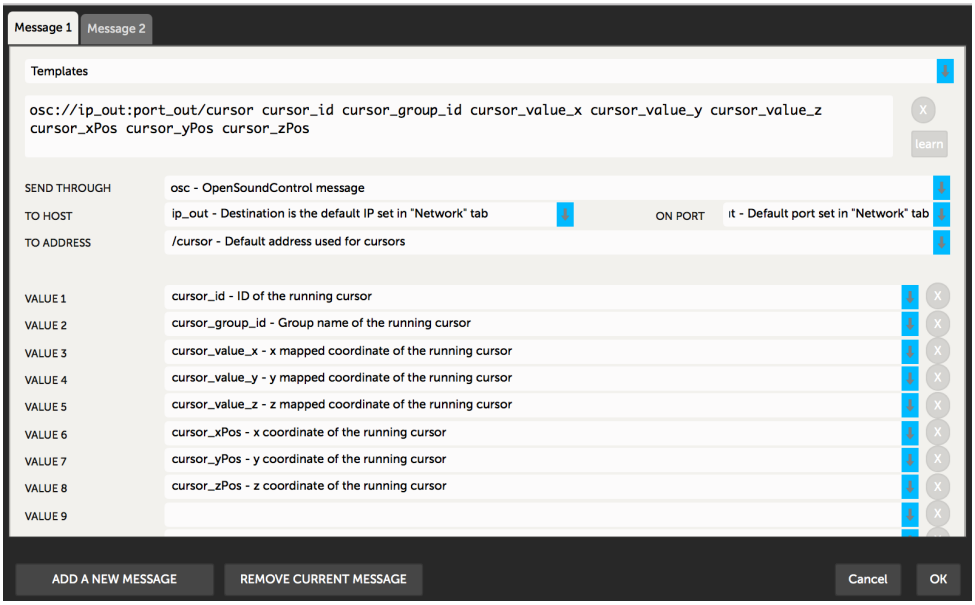
Example

The following example will be use Iannix (www.iannix.org) a free OSC generating graphic score generator.

First of all, set the Host Iannix IP to the Paca(rana) and port 8000 (as described above).

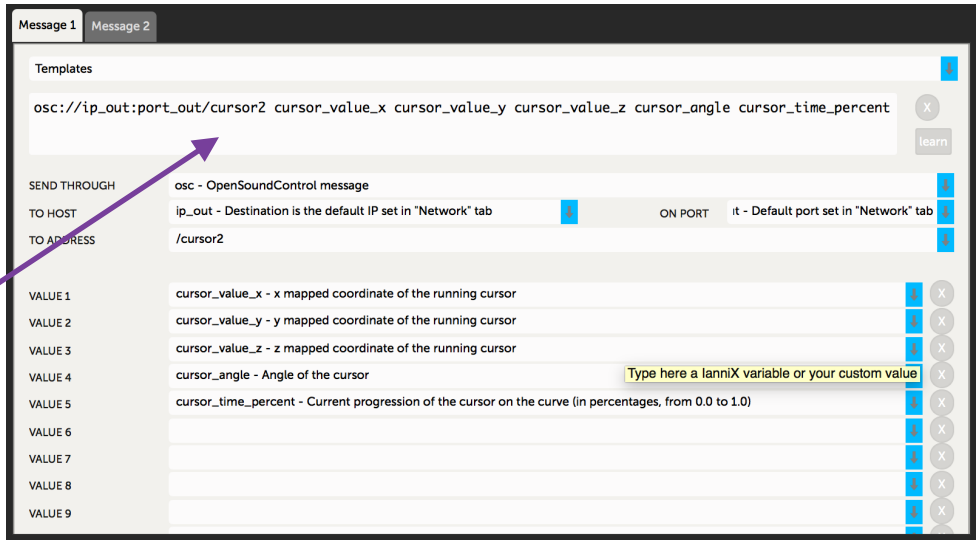
In Iannix you add a cursor which will travel around a complex 3D curve. It can output a lot of data about its path around the curve, such as position, angle, distance travelled.

Create a curve and a cursor to travel around it. From the OBJECTS tab in the inspector, select the cursor. Go to INFOS tab, and select EDIT MESSAGES from the MESSAGES sub tab in the Inspector.



This is the default set of messages Iannix sets up for you. Kyma cannot understand Strings, so we don't need Group Name. Also, we don't need the cursorID.

A more useful message to construct for use with ReadOSC in Kyma would be the following. Notice that I have created a custom OSC address stem `/cursor2`.



In OSCMessages field you will read this message, which is one message with an address stem called `/cursor2` with 5 float arguments.

In readOSC, you read it like this: [2]

```
'/cursor2,fffff'
```

ReadOSC will generate 5 (hidden) Kyma syntax receiver widgets, one for each float argument.

`!osc_cursor2__1 !osc_cursor2__2` etc. <--- generated by ReadOSC and hidden.

You can now choose more meaningful names for the values generated in Iannix by `/cursor2`.

GeneratedEvents

The incoming OSC messages are received using special Kyma syntax. These raw widgets will be hidden from you, because you don't need to see them when using ReadOSC.

Instead, you can remap them to more meaningfully named generated events, to use as sound parameters throughout your signal flow.

For example:

```
!IannixCursor2.x
!IannixCursor2.y
!IannixCursor2.z
!IannixCursor2.angle
!IannixCursor2.progress
```

Make sure that your chosen names are not being generated anywhere else in your Kyma signal flow, Multigrid or Timeline.

You must have the same amount of generated events as there are messages coming in (which will be defined by the message list you input at the OSCMessages parameter field (see the help description)

You cannot perform any Capytalk transformations here. Enter `!HotValues` with exclamation marks, separated by a new line or space. The syntax colouring will be red, if you have done it correctly. You can even add notes here in double quotes to remind you what things are.

---

[2] Kyma might ask you to make it clear if this is an Element or the entire array of the OSCMessages field, when there is only one message in there. Choose Element in almost all cases, except when you are building the array algorithmically using a Smalltalk `collect:` expression.

`SignalMinValues SignalMaxValues`
`OutputMinValues OutputMaxValues`

The input output matrix. Use these four Input/Ouput arrays to remap the range of each incoming OSC message to a different range for your Kyma Sound design. They must be a list of constants, no Capytalk is allowed here, but you can use Smalltalk code to construct or alter the list of values at compile time.

TIP: If there are not enough remapping values to match each generated argument and message, then the last value will be repeated.

Examples:

Incoming value is scaled as an angle and you need it is a full range normalised signal.
InputMin: 0
InputMax: 360
OutputMin: -1
OutputMax: 1

Invert and scale:
InputMin: 0
InputMax: 255
OutputMin: -1
OutputMax: 1

Map to a set pitch range:
InputMin: -1
InputMax: 1
OutputMin: 60
OutputMax: 72

Algorithmic constructor generates linear distribution for each argument and voice:

```
{ 1 to: 5 collect: [ :v | (1/?NumberVoices) * ?VoiceNumber ] }
```

`StealthMode`

When you have finished routing and are happy with the OSC configuration, go into stealth mode to hide all OSC receiving widgets and generated events from the VCS.

They will continue to broadcast their values through the GeneratedEvent names you have specified, so you can refer to them by those names elsewhere in your sound design.

For example, map the `!IannixCursor.x` to the formant of an oscillator and the `!IannixCursor.y` to frequency.

Smooth

Realtime smoothing for averaging out the OSC input over a duration.
The smoothing also acts on any changes to the dynamic Scaling (see parameter description for Scale).

Scale

Further dynamic scaling can be applied to each generated event, after remapping through the input output matrix. Accepts an array of hotvalues. Expressions must be enclosed in curlies.

TIP: If there are not enough scaling elements for each message and argument, the last element will be repeated

Example:

Use the Scaling array to fade in or out the amount of modulation coming from a particular OSC message.

```
1 1 {!FadeUpSensorInput ramp00: 30 s} {!FadeUpSirenLFO ramp00: 60 s}
```

Or shrink, blink, modulate and expand all values at once with one expression, respecting the scaling matrix you defined.

```
{!GlitchEffect * (0.1 s random)}
```

PauseIncomingOSC

A positive value here will activate the receiving widgets.
A zero will stop listening to the OSC messages specified in this instance of ReadOSC.

NeverEngineLabs OSCtoSound: Class and Parameter Description

OSCtoSound takes a single message in standard OSC syntax and converts it into a GeneratedEvent as well as outputting it as audio stream. Using the Input/Output arrays makes it easy to remap the incoming OSC message in a meaningful way. In StealthMode you can even hide the OSC widgets created and keep a nice and clear VCS. You can process the audio stream as you would any other audio signal in Kyma. You could even record it to disk.

Some tips for happy OSCtoSound:

- You must adhere to the OSC Syntax in the OSCMessage field (see parameter help for OSCMessage)

- Make sure your device is sending the OSC data using port 8000 and your Paca(rana)'s IP (Configure -> OSC)

- Make sure your Paca(rana) and the sending device are connected to the same network. Use the Paca(rana)'s ethernet port on the back to connect it to either a router or directly to the device (most likely your computer)

- When choosing a name for the OSC controlled GeneratedEvent, you must be **absolutely certain** that the name does **not already exist as a generated event** elsewhere in your signal flow, multigrid and Timeline, or you risk a conflict that could crash Kyma.

OSCMessage

OSCtoSound is a mono receiver. It only accepts one message and one argument. The message must be completed by a comma followed by the symbol (f = float) for one argument.

```
'/anIncomingMessage,f'
```

GeneratedEvent

Enter an EventValue name (including the exclamation point prefix) for the generated event after remapping of the incoming OSC message. You cannot do any further Capytalk expressions here, it is only for declaring the name.

Silent

Uncheck silent to also render the incoming OSC signal as a sample rate audio signal. If you wish to work with this method, make sure you are remapping the incoming OSC range to a (-1,1) range, as this is the full range of an audio signal in Kyma.

Scale

Scale the input before remapping. Accepts hot values or pasted sounds.

InputMin InputMax OutputMin OutputMax

Fixed constant only. Remap the range of the message at the OSC input to a different range for Kyma.

Examples:
Incoming range is scaled as an angle and you need it is a full range normalised signal.
InputMin: 0
InputMax: 360
OutputMin: -1
OutputMax: 1

Invert and scale the osc coming in:
InputMin: 0
InputMax: 255
OutputMin: 1
OutputMax: -1

Smooth

Convenient smooth on the incoming signal with a duration in seconds. Set to 0 to bypass smoothing.

Interpolation

Turns interpolation on and off. Useful to upsample the audio output to samplerate.

StealthMode

When you have finished routing and are happy with the OSC configuration, go into stealth mode to hide the OSC receiving widget and generated event from the VCS. They will continue to broadcast their values through the GeneratedEvent name you have specified, so you can refer to them by those names elsewhere in your sound design.

PauseIncomingOSC

A positive value here will activate the receiving widget. A zero will pause it holding the last value received.

InitialWidgetType

This option lets you decide what type of Widgets will be created the first time this instance of OSCtoSound is run (and after coming out of stealth mode). Its handy so as you don't fill up the VCS with many widgets.

NeverEngineLabs SoundToOSC: Class and Parameter Description

SoundToOSC takes an audio input and transmits it to a single message destination you define with standard OSC syntax, and also to a rewritten GeneratedEvent. You could use this mono send in many creative ways, for example as a method to replay OSC derived from previously captured OSC gestures, recorded to disk through OSCtoSound. You could even add echo effects and more before converting over to the OSC destination message.

Some tips for a happy sending...

- Check you have set the correct IP and Port to reach the receiver from Kyma (see parameter help for SendToIP)
- You must adhere to the OSC Syntax in the `OSCMessages` field (see parameter help for `OSCMessages`)
- You must have a signal associated to each OSC message AND each argument in the case that a message has more than one argument.
- If `Rewrite` is active, then you must also have a `RewrittenEventName` defined for each message **and each argument**.
- **IMPORTANT**: When choosing new names in `RewrittenEventNames` take care that those names are not used elsewhere in the signal flow, grid or timeline, or there could be an internal conflict in Kyma and may cause a crash.
- In stealth mode you can't have a completely empty VCS. You will need to have at least one widget for background OSC to keep streaming.
- A good tool for monitoring OSC network traffic is essential for debugging and programming. Kyma has Event log available from DSP Status/Configure OSC. Camille Trouillard's OSCulator is very Kyma friendly. The automation sequencer Iannix has a good built in message log too.

`OSCMessage`

OSCtoSound is a mono sender. It only accepts sends message and one argument. The message must be completed by a comma followed by the symbol (f = float) for one argument.

```
'/anIncomingMessage,f'
```

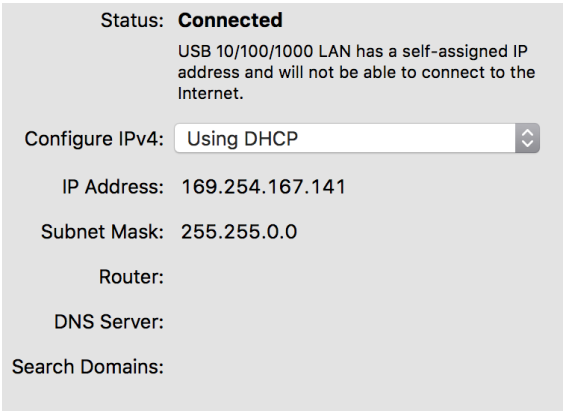
RewrittenEventName

Enter an EventValue name (including the exclamation point prefix) for the generated local event after remapping of the outgoing OSC message. You cannot do any further Capytalk expressions here, it is only for declaring the name. Make sure it is not the same name as another GeneratedEvent somewhere else in the Signal flow.

SendToIP

Here you put in the address of the device you wish to send to.

The Paca(rana) must be connected to the same network as this IP with a Cat5 ethernet cable from the ETHERNET port on the back. Make sure you haven't plugged it into the EXPANSION sockets by mistake!



TIP: This IP is not the same as the one you get from the Kyma DSP Status info.

In Mac OS, the correct IP to use is the one you will see in System Preferences/Network. The service will be an ethernet network and should say CONNECTED in bold, and will usually have a status of self-assigned IP. [3]

SendToPort

This is the Port number to send the signals to. Software or a device with the IP specified in the `SendToIP` parameter field, listening on this port number, will receive the Kyma signals through the `OSCMessage` you have defined.

Smoothing

Convenient smooth on all the outgoing signals with a duration in seconds.
Set to 0 to bypass smoothing.

PollingTrigger

Messages are only sent when this trigger changes from 0 to positive. Use it to reduce data over the network if necessary... `25 ms tick` is usually a pretty reliable polling rate. Alternatively use it as a way of muting or pausing the outputting of all messages specified in this instance of SendOSC. For example: `25 ms tick * !mute`

---

[3] Read more about ports, IP and Kyma at http://kyma.symbolicsound.com/qa/1895/how-do-i-change-the-sending-port-for-osc

StealthMode

When you have finished routing and are happy with the OSC configuration, go into stealth mode to hide all OSC controlling generated widgets from the VCS.

They will continue to broadcast their values to the `GeneratedEvent` you have specified AND transparently to the OSC you have specified. Even though they are hidden, you can still refer elsewhere in your sound design to the signals being sent out over OSC by using the more meaningful names that you defined in `GeneratedEvent` .

For example, map something like `!VirtualObjectSize` to an object in a virtual world, a filter cut off and a reverb decay at the same time, whilst keeping your VCS clutter-free.

TIP: In stealth mode you can't have a completely empty VCS. You will need to have at least one widget for background OSC to keep streaming.

InitialWidgetType

This option lets you decide what type of Widgets will be created the first time the Sound is run. Its handy so as you don't fill up the VCS with many widgets.

Each widget will remember its display type after the first time it is created, so you will have to change the type by hand in the VCS after that.

1 - SmallFader (only numbers)
2 - Panpot
3 - Gate
4 - Fader

Known bugs in v1.0 of OSC Tools

When in stealth mode (in Mac OS X) if Kyma is not the frontmost application, the OSC data will be throttled down to about 1 s update. Workaround is to keep Kyma always as the active application when in Stealth mode, or to avoid using stealth mode and work with visible OSC generating widgets.

Sometimes, if entering a single OSC message in the SendOSC or ReceiveOSC, Kyma will need the user to specify if the data input is an ELEMENT or the entire ARRAY. Usually you would just choose ELEMENT unless you are generating the Array using SmallTalk in line scripting.


NEVERENGINE LABS™