NeverEngineLabs

ReadOSC (2024)

ReadOSC takes a list of messages in standard OSC syntax and converts them into Kyma GeneratedEvents. The new 2024 version features dynamic min/max remapping of incoming values with wrap-around normalisation to (0 , 1) or (-1 , 1) ranges. Definitely more suitable for general use in Kyma than the ranges you might find coming in from an external system!

Basic Usage

Listen for OSC message:

```
▶ '/SrcPos/1/xy,ff' <--- the comma ff means this message has two arguments
```

Kyma syntax: *created by ReadOSC behind the scenes*

```
'!osc_srcpos_1_xy__1'
'!osc_srcpos_1_xy__2'
```

Rewritten GeneratedEvents:

```
!helicopter.x
!helicopter.y
```

By adding .x and .y to the end of each name, Kyma will automatically make a 2D widget.

Some tips for happy ReadOSC:

- 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). Read more about ports, IP, and Kyma at here

Credits

Class design and programming by Cristian Vogel NeverEngineLabs 2024
ReadOSC uses code from OSCHelper Tool by Symbolic Sound.
Support hotline: support@neverenginelabs.com

Parameter Guide

Enable

Enable the OSC widgets and outputs, leave this on, but it may be useful for debugging?

OSCMessages

Enter list of OSC messages one on each line, enclosed by single quotes.
Each message much 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 variables into your OSCMessages. For example:
▶ **`'/lamp/'&?VoiceNumber&'/brightness,f'`**

GeneratedEvents

The incoming OSC messages are received using the special Kyma renaming protocol. These raw widgets are hidden from you, because you don't need to see them.
Instead, you are encouraged to come up with more meaningfully names for the incoming generated events, to better work with sound throughout your Kyma signal graph.

> Conveniently, the 2024 version of ReadOSC also has a powerful normalisation feature, which is applied to incoming ranges, whatever they may be , which gives the composer far more flexibility when working with arbitrary numerical ranges being sent from outside Kyma.

RULES:

1) You must have the same count of generated events as there are messages coming in (which will be defined by the message list you input at the **`OSCMessages`** parameter field.

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

An example:

Iannix (the excellent open source graphical spatial composer) sends messages made of one stem and many float arguments.

In your **`OSCMessages`** field you might connect a cursor sending its x and y location like this:

```
'/cursor,ff'
```

This will generate two events in Kyma, one for each value (the x and the y). These received variables can be re-written as nicely named generated events, using any naming schema you like. For example:

```
!IannixCursor.x
!IannixCursor.y
```

(In Kyma adding **`.x`** and **`.y`** will automatically generate a 2D fader yo)

Input Transformation: Normalisation and Offset

ReadOSC 2024 edition introduces dynamic incoming data transformation. The value of message being listened for can have a fixed or dynamic minimum, maximum and offset value, which will always be normalised to Kyma's preferred (0 , 1) or (-1 , 1) range. This makes it easy to co-relate incoming parameter ranges when you know what they are going to be sending in terms of minimum value and maximum value. For example, angles are often coming in as degrees, (-360 , 360). The offset array is also a useful tool, as it can be used to shift signals around. Things can get interesting with realtime alterations of incoming OSC, they will always stay normalised as I have implemented a wrap-around algorithm, so if you offset past an expected maximum instead of clipping, the range will wrap around. You can experiment with these features creatively in your sound designs to jazz up and coming signals for use in Kyma.

The following fields are Arrays, so the usual rules apply. Capytalk must be in enclosed in curlies, and the last element repeats if the Array has less elements than it needs.

InputMinimums

An array that can be used to dynamically normalize for Kyma from the expected minimum values of each incoming OSC value. If there are less values in this Array than there are generated events, the last value will continue to be applied.

Example 4 incoming messages have been defined.

```
0 0 {!MinAngle smoothFor: 2 s} -1
```

Would mean the first two values have an expected minimum value of 0 , but the third incoming value can have its expected minimum altered dynamically, and for message values after that will all have a fixed expected minimum of -1.

InputMaximums

An array that can be used to dynamically normalize for Kyma from the expected maximum values of each incoming OSC value. If there are less values in this Array than there are generated events, the last value will continue to be applied.

InputOffsets

An array that can be used to dynamically offset each incoming OSC value whilst keeping it normalised for Kyma. There's a really useful wraparound implementation, instead of clipping at min and max limits. As before, If there are less values in this Array than there are generated events, the last value will continue to be applied.

Example any number of incoming messages have been defined.

```
{!offsetAll smoothFor: 3 s}
```

Would mean the all values will be offsettable with only one smoothed fader.

Normalise01

A positive value here will perform the receiving normalization for Kyma to a (0,1) range. If negative or zero, then a (-1,1) range will be calculated.

Smooth

Real-time smoothing for averaging out the OSC input over a duration.

PauseIncomingOSC

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

InitialWidgetType

This option lets you decide what type of Widgets will be created the first time this instance of **`OSCtoGlobalControllers`** is run (and after coming out of stealth mode)
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

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.

NeverEngineLabs.

SendOSC (2024)

`SendOSC` is a tool that takes a list of destination messages in standard OSC syntax and converts them into OSC generating widgets that Kyma can control. These widgets can be used to control a receiving software or device on the same network as the Paca(rana). The outgoing OSC widgets in the VCS will be mirrored and rewritten with your custom naming schemas for use as parameters in the rest of your Kyma sound design.

Basic Example

Original OSC message:

```
'/SrcPos/1/xy,ff'
```

The **`,ff`** means this message has two float numerical arguments. At least one argument is obligatory.

In Kyma default syntax

```
!osc_srcpos_1_xy__2
!osc_srcpos_1_xy__2
```

SendOSC rewrites these messages by default to more meaningful names.

Rewrite to:

```
!helicopterSound.x
!helicopterSound.y
```

⇧

Tip! **Adding .x and .y will automatically create a 2D widget!**

Now you can control sound design and an external spatialisation system with the same Event Expressions all from Kyma with nothing in between. Joy of Joys!

Tips for Happy SendOSC

- Check the correct setting of IP and Port to reach the receiver from Kyma In most common cases, the IP you should use is the IP of the Mac/Windows connection to the Paca. Find it in your system settings.
- Adhere to the OSC Syntax in the **`OSCMessages`** field (see parameter help for **`OSCMessages`**).
- Ensure a signal is associated with each OSC message and each argument (if a message has more than one argument).
- If 'Rewrite' is active, define an **`EventName`** for each message and argument.
- Avoid conflicts when choosing names for **`ExtraGeneratedEvents`**.
- In stealth mode, you can't have a completely empty VCS; keep at least one widget for background OSC to keep streaming.
- Monitor OSC network traffic for debugging and programming (e.g., OSCulator or Max).

Class Design and Programming

Class design and programming by Cristián Vogel for NeverEngineLabs 2024 v24.1.b1. **`SignalsToOSC`** uses code from OSCHelper Tool by Symbolic Sound.
Support hotline: support@neverenginelabs.com

Parameter Guide

Enable

Enable the OSC widgets and outputs. Generally leave this on, but it may be useful for debugging.

SendToIP

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. In most cases, this is the IP from System Preferences/Network interface that is connected to the Paca APU. Read more about ports, IP, and Kyma here(http://kyma.symbolicsound.com/qa/1895/how-do-i-change-the-sending-port-for-osc).

RewrittenEventNames

In Kyma, outgoing OSC controller messages are transmitted via the Virtual Control Surface, and adhere to the Kyma OSC syntax which looks like this example.
`!osc_virtualSource_pos__1`

Rewriting using SendOSC allows more meaningfully named generated events, which can be easily re-used as parameters throughout your signal flow. This encourages the composer to use values that route to an external system in other aspects of a sound design being simultaneously. For example, using a re-written name for a common spatial parameter like *distance from centre*, means you could implement your own perceptual filtering on Kyma a audio stream which relates to spatial features of a virtual object emitting that audio stream. **This is a powerful workflow.**

Rewrite

When active, duplicates of the Kyma syntax OSC widgets are created using the **`GeneratedEvent`** names defined in the **`ReWrittenEventNames`** field as described above. Nevertheless it is still possible to disable the creation of rewritten VCS widgets to work with raw Kyma OSC syntax widgets.

OutputScaling

An optional array that will scale the outgoing OSC values by any amount. If there are less values here than there are outgoing messages, the last value in the OutputScalling array will be applied to everything following it.

So for example if you had 8 messages you could control the last four all at once with the same Capytalk, like this;

```
▶ 1 1 1 1 {!FadeTopQuad smooth: 3 s}
```

If you had 8 messages and you want to scale the 4th and the 6th, it is required to pad this array. In such case, you could try:

```
▶ 1 1 1 !Scale4 1 !Scale6 1 1
```

alternatively, generate as many as you need

```
▶  ( !Scale copies: 4 )
```

Any Capytalk must be in curlies, like always in Kyma for example:

```
▶ {(1 repeatingRamp: 30s) * 360} { !GoFadeUp smooth: 20 s }
```

And remember, by default you will only observe the scaled signals happening at the OSC receiver, not in the VCS.

InitialWidgetType

Decide what type of Widgets will be created the first time the Sound is run. Each widget will remember its display type after the first time it is created.

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

SendToPort

Port number to send the signals to. The software or device with the specified IP will receive the Kyma signals through the **OSCMessages** you have defined.

OSCMessages

Enter a 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.

Here are some examples;

```
'/cursor/1/xy/,ff'

 "use Smalltalk to splice something at build time"
'/myObject/'&?VoiceNumber&'/xyz/,fff'
```

Signals

Array of CapyTalk, hotvalues, constants or pasted Sounds that will be transmitted as each OSC message value. Enclose Capytalk expressions within curly braces.

PollingTrigger

Messages are sent only when this trigger changes from 0 to positive. Use it to reduce data over the network if necessary, or perhaps to send some initialisation settings.

Smooth

Convenient smooth on all outgoing signals with a duration in seconds.

StealthMode

Hide all OSC controlling generated widgets from the VCS. Continue broadcasting values to the **`RewrittenEventNames`** specified and transparently to the specified OSC.