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.