



osculator

application version 2.12 — manual version 20120515

# Table of contents

Foreword	4
▶ Acknowledgments	
Introduction to OSC	5
▶ A gentle introduction to IP networking	
▶ UDP and TCP	
▶ The OSC message in detail	
The User Interface	11
▶ Main window	
▶ Scalings page	
▶ Notepad	
▶ Keycode helper	
▶ Quick Look	
▶ Operations on messages	
Preferences	19
Parameters window	21
▶ Key Combos	
▶ AppleScript	
▶ I/O	
Presets	27
▶ Concept	
▶ Managing the presets	
▶ Preset change actions	
▶ Importing presets from other files	
Events	29
▶ Overview	
MIDI output	32
▶ MIDI CC	
▶ MIDI CC Toggle	
▶ MIDI Prg	
▶ MIDI Notes and related events	
OSC Routing	36
▶ Overview	
▶ OSC Routing parameters	
▶ OSC Routing Edit Window	

▶ Using Variables	
 Virtual Devices	42
▶ Keycode	
▶ Key Combo	
▶ Mouse	
▶ HID	
 Meta events	45
▶ Enable, Enable Toggle & Enable Combine	
▶ Presets	
▶ Change Channel	
▶ Variables	
 MIDI input	47
▶ Received messages	
▶ How to use the /midi/note/<channel> message in practice?	
▶ Automatic back-mapping in practice: Ableton Live configuration	
 Wiimote	51
▶ Support	
▶ Connecting the Wiimote	
▶ Connecting an extension	
▶ Configuring the Wiimote	
▶ OSC messages	
▶ Advanced control	
 Space Navigator	58
▶ Overview	
▶ Space Navigator messages	
▶ Calibration	
 TUO	59
▶ TUO enabled software	
▶ TUO messages	
 Wacom Tablet	61
▶ Quick Start	
▶ Wacom tablet messages	
 Appendices	63
▶ Event input range	
▶ Event triggering	
 Legal & Copy Notices	68

# Foreword

Personal computers have played a key role in the creation of sound, images, and multi-media experiences for many years now, both for professionals and amateurs. The revolution in sensor miniaturisation and cost in the past few years has led to the increasing use of game controllers, touch-enabled surfaces and more generally wired and wireless remote controllers of all sorts as inputs to digital creation. Accompanied by a massive increase in cheaply available computer processing power<sup>1</sup>, such devices have become widely available and affordable. Because these controllers are cheap or easily found in stores, a whole new class of creative possibilities has emerged, rendering interaction with the computer and its surrounding devices much more tangible.

There is however a problem with the growing number of solutions: complexity. Existing creative software, for example DAWs, do not support most of this new class of controllers out of the box, precluding novices from benefitting, while advanced users have to get over a steep learning curve or cobble together makeshift solutions for each new controller.

OSCulator is software designed to solve this problem by connecting a comprehensive and growing list of controllers with production tools. OSCulator is powerful and flexible, yet user-friendly.

- ▶ It solves simple problems with almost no configuration.
- ▶ It can scale up to control a distributed network of computers as well, such as in large art installations.
- ▶ It doesn't have a steep learning curve, because the same concepts are consistently applied throughout.
- ▶ It is actively developed and supported, and has an ever-growing user base.

## Acknowledgments

Thanks to Manu Iyengar and Lenny (Somascape) for their kind support during the writing of this manual.



---

<sup>1</sup> with a processing power comparable to the 4 processors version of Cray 2, the iPad 2 [could have stayed](#) on the list of the world's fastest supercomputers through 1994!

# Introduction to OSC

[Open Sound Control \(OSC\)](#) is a protocol for communication among computers, sound synthesisers, and other multimedia devices that is lightweight and leverages current networking technologies. As an attempt to simplify the vast problem of inter-connectivity, OSC has been chosen as the basis for OSCulator.

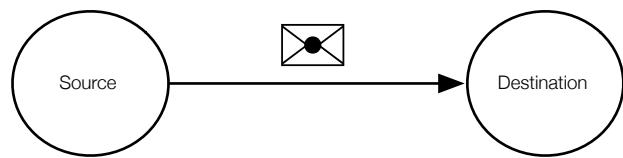
Understanding how OSC works is recommended, although not essential. This chapter is not meant to be an exhaustive description of the IP and OSC protocols, but can serve handy at the time more in-depth knowledge is desired.

## A gentle introduction to IP networking

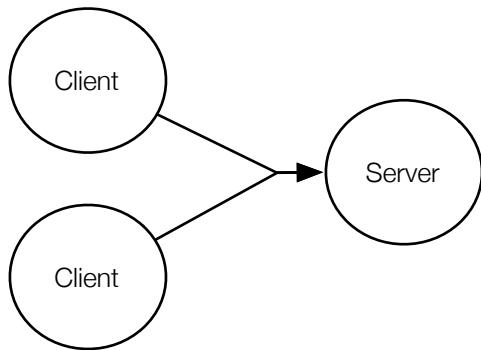
Although the OSC specification does not enforce any particular type of transport, OSC is nowadays mostly used over traditional networks known as IP (Internet Protocol). The following section will try to explain in simple terms how an IP network works with a special focus on OSC.

### Messages

A message is a unit of information sent from a source to a destination. In the OSC protocol, messages are used most of the time to describe a change, but they can also be used to describe more complex things<sup>2</sup>.



*A source sends a message to a destination*



*Two clients sending messages to one server*

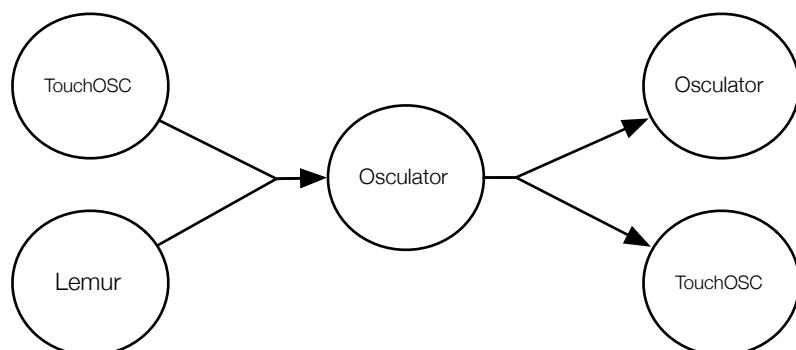
### Clients and Servers

In OSCulator, the OSC protocol uses the IP network<sup>3</sup> to carry messages from a source to a destination. Sources of OSC messages are usually called *OSC Clients*, and destinations *OSC Servers*. One client can send OSC messages to only one server at a time, but one server can receive from many clients.

<sup>2</sup> e.g. the TUIO protocol is built on OSC and is used to describe multi-touch events.

<sup>3</sup> see next section

Here is a possible example of the same graph with actual devices:

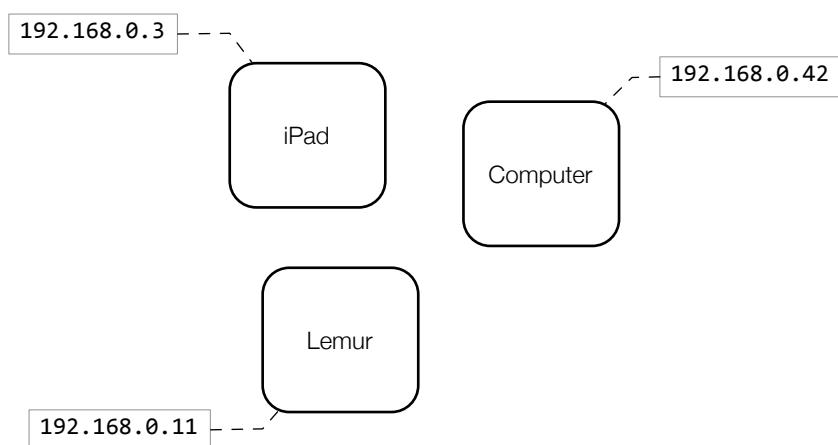


*Two devices sending messages to Osculator,  
that is itself messaging two other clients*

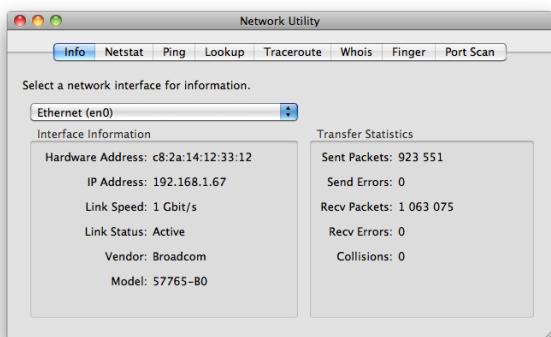
It should be noted that OSCulator is both able to receive (as a server), and send to several destinations (as multiple clients).

## IP addresses and host names

The IP network uses *addresses* to represent the location of computers, also known as hosts. An address is written as a dotted quadruplet, for example: 192.168.0.1.



*Devices on an IP network have an IP address that is used to find their location, like a telephone number*



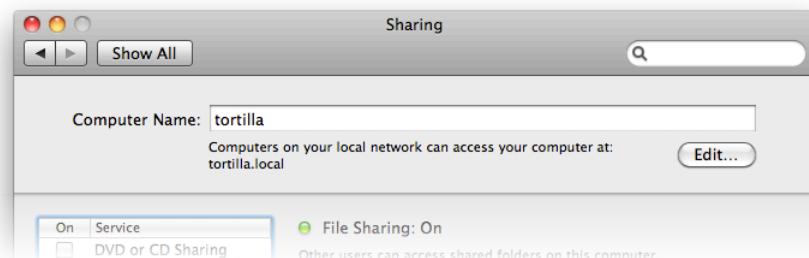
*Network Utility reports useful information about the network, and is located in /Applications/Utilities*

On OS X, the IP address of a computer can be obtained using the *Network Utility*, and changed using the *Network Preference Pane*.

The IP address is a very basic element, and because it is not easy to remember, computers usually have an associated name. For example, to access Twitter, your computer first has to find the IP address of Twitter's web server, a little bit like when you search for a phone number in an address book.

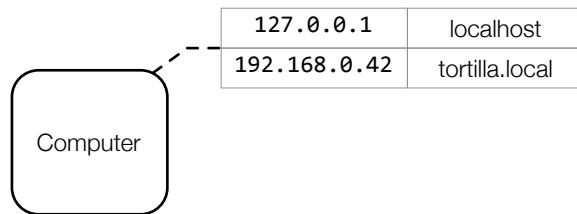
The name of this server is [www.twitter.com](http://www.twitter.com), and has the IP address **199.59.148.10**.

On OS X, you can find and change the network name of your computer in the Sharing Preference Pane:



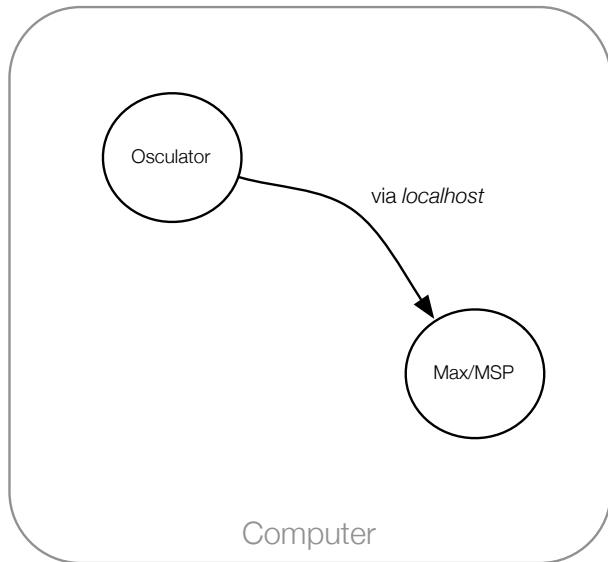
*The Sharing Preference Pane is used to change the computer name to something we can remember*

A computer can reference itself by using a special address called the *loopback address*. This address has a name as well: *localhost*.



*A computer has a local address 127.0.0.1, and each address can have names.*

If you want to send a message between OSCulator and Max/MSP for example, and both programs are running on the same computer, you must use *localhost* as the name of the host you want to send the messages to. Here, your computer will act both as a client (OSCulator) and a server (Max/MSP).

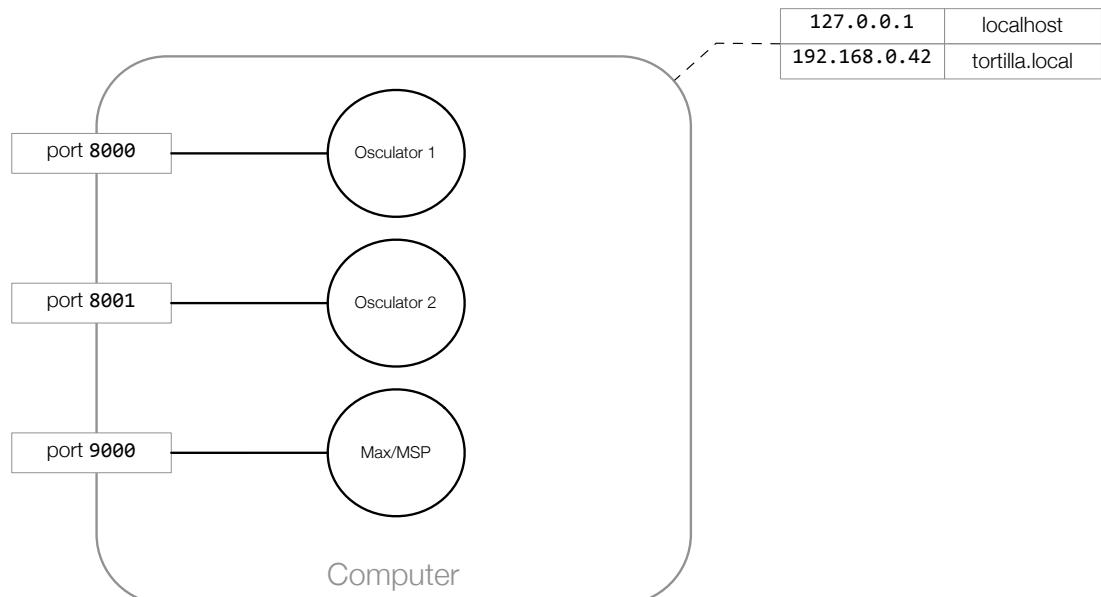


*Viewed from the inside, a computer uses the name localhost to reference itself in the network*

## Ports

By definition, the destination of OSC messages are OSC servers. Just as computers are identified by their IP address or name, services are using network ports. A port is like a door to a service in a computer.

Two services can not share the same port number.



*Ports are like doors to a service in a computer.  
The computer is referenced by its address, and the service by a port number.*

## Bonjour



[Bonjour](#) is a service discovery protocol. It locates services such as TouchOSC running on an iPad, or OSCulator running on another computer without the need for any configuration.

Devices are becoming increasingly mobile, and the network more and more dynamic. Bonjour alleviates tedious and complex setups by making application independent of the low-level configuration aspect. In Bonjour, a service is simply represented as a name, usually automatically given after the computer's name.

As long as two computers are connected together by an Ethernet cable or a Wi-Fi ad-hoc connection<sup>4</sup>, they will be able to find and use their respective services.

## UDP and TCP

UDP and TCP<sup>5</sup> are network protocols used for communicating OSC messages between two devices. OSCulator supports both protocols for transmitting and receiving OSC messages.

UDP is the most widely used at the moment, but you can opt to use TCP to communicate with another device if supported.

The major differences between TCP and UDP are the following:

- ▶ OSC messages sent over TCP are guaranteed to be received in the order they have been transmitted (not for UDP).
- ▶ On a reliable network link, an OSC message TCP is guaranteed to be transmitted (not for UDP).
- ▶ TCP has more latency than UDP, however that should not be an issue for most uses.

When a OSCulator document is opened, the OSC server listens for both protocols on the same port. So for example if the input port is 8000, OSCulator will listen for OSC messages on UDP port 8000 and TCP port 8000.

For configuring the transport protocol when sending OSC messages, see the [OSC Routing](#) chapter.

## The OSC message in detail

An OSC message is made of:

- ▶ an *address*  
Used as an identifier for the message.
- ▶ a list of *arguments*  
Each argument can be of different kind: numbers, strings, etc.

Here are some examples of possible OSC messages:

A message with one argument:

A message with a string and 3 integers:

A message with two floats:

When OSCulator triggers an event, it can either use a message or the argument of a message as the input. Because many

<sup>4</sup> computer-to-computer

<sup>5</sup> see [http://en.wikipedia.org/wiki/User\\_Datagram\\_Protocol](http://en.wikipedia.org/wiki/User_Datagram_Protocol) and [http://en.wikipedia.org/wiki/Transmission\\_Control\\_Protocol](http://en.wikipedia.org/wiki/Transmission_Control_Protocol)

OSC messages have only one argument, novice users often make the confusion between the message itself and its arguments.

Here is an example of an OSC message that OSCulator received. This message has two arguments, numbered as 0 and 1. We can see that there are three lines available for configuring an event:

	Message	Event Type	Value	Chan.	↔
<input checked="" type="checkbox"/>	▼ /3/xy	-	▲ -	▲ -	▲ -
<input checked="" type="checkbox"/>	0	-	▲ -	▲ -	▲ -
<input checked="" type="checkbox"/>	1	-	▲ -	▲ -	▲ -

The first line is used for events that can process the message with all of its arguments. For example, we can choose to send this message to another computer, or print it for debugging:

	Message	Event Type	Value	Chan.	↔
<input checked="" type="checkbox"/>	▼ /3/xy	OSC Routing	2-Computer ➔ /somewhere/else	2 -	2 -
<input checked="" type="checkbox"/>	0	-	▲ -	▲ -	▲ -
<input checked="" type="checkbox"/>	1	-	▲ -	▲ -	▲ -

Another possibility would have been to use the value of the first argument and make it a MIDI message instead:

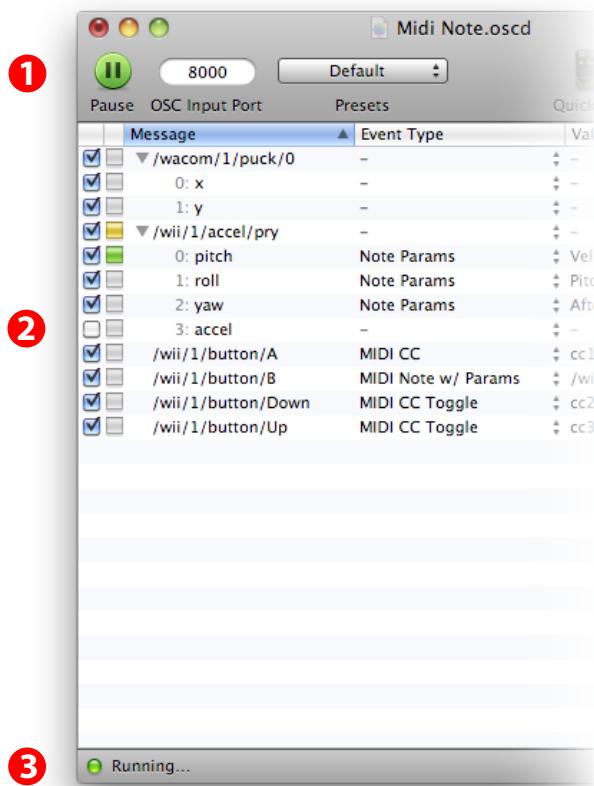
	Message	Event Type	Value	Chan.	↔
<input checked="" type="checkbox"/>	▼ /3/xy	-	▲ -	▲ -	▲ -
<input checked="" type="checkbox"/>	0	MIDI CC	▲ 7	▲ 1	▲ 1
<input checked="" type="checkbox"/>	1	-	▲ -	▲ -	▲ -



# The User Interface

## Main window

Most of what happens in OSCulator is located in the Main Window. This window displays the list of messages that OSCulator has received so far and their associated settings. The changes made in the Main Window, can be saved to a file, generally referred to as the *document*.



### 1. Toolbar

A toolbar with customisable items. The default layout suggests:

- a **Start / Pause** button
- the **OSC input** port number
- the **Presets** menu
- the **Quick Look** viewer, useful to have a visual cue of the input
- the **Parameters window** button, giving access to parameters

### 2. Messages list

The table showing the list of registered OSC messages.

### 3. Status bar

A blinking monitor indicates the running status, and a label displays messages of interest to the user.

Message	Event Type	Value	Chan.
/wii/1/accel/pry	Note Params	Velocity	6
0: pitch	Note Params	Pitch	5
1: roll	Note Params	Aftertouch	
2: yaw	Note Params		
3: accel	-	-	
/wii/1/button/A	MIDI CC	cc1	1
/wii/1/button/B	MIDI Note w/ Params	/wii/1/accel/pry	1
/wii/1/button/Down	MIDI CC Toggle	cc2	1
/wii/1/button/Up	MIDI CC Toggle	cc3	1

#### 1. Enable

A checkbox to enable or disable the event. If the checkbox is unchecked, the associated event will not be triggered.

#### 2. Activity

An activity monitor that blinks when a value is changing:

- yellow when nothing is performed,
- green when an event is triggered correctly,
- red if an error occurs.

This column is also sensitive to mouse clicks: clicking the activity monitor will send a default message, useful for testing if an event is correctly configured.

#### 3. Message Address

This is the name of the registered message. When the disclosure triangle is down, this column displays the indices of the arguments, or their user-friendly memo if they are available. You can edit these memos by double clicking the row.

#### 4. Event Type

A drop-down menu that shows the list of all Event Types. If you want to assign an event to a message, use this menu first, then go to the Value menu.

#### 5. Value

A drop-down menu that shows the list of all possible values depending on the chosen Event Type.

#### 6. Channel

MIDI related Event Types use a channel (such as MIDI CC, Kyma CC, or MIDI Note).

When using a HID event, the channel is the number ID of the joystick (two virtual joysticks are available).

## Creating a new document

To create a new document, choose *File → New*, or press *⌘N*. A new document is empty, because no message has been received yet.

In order to populate this table, you will need to send OSC messages to OSCulator, or connect a Wiimote or a SpaceNavigator. This has two benefits: test whether the connection is working correctly, and automatically learn the OSC message.

## Filtering messages by name

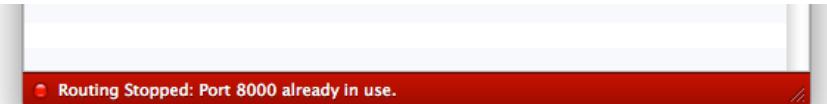
The toolbar includes a search field for filtering messages by name. To activate filtering, press *⌥⌘F* and type a search string.

## Multiple documents

Multiple documents can be opened at once. Each document is then represented by an individual main window, each of them having an OSC server is listening on different ports.

This is useful when you want to split functionality across several files. For example, if you need to work with two different devices, you must separate their configuration using two different files. Though you can mix both configurations in the same file, it is easier and less confusing to manage things this way.

Two windows can not share the same OSC input port. If two windows have the same port, then one of the two windows will be not able to start the OSC engine. In this case, an error message is displayed, and the status LED turns to red:



## Scalings page

When working with foreign controllers, or in order to adjust the sensibility of a sensor, it is desired to specify the range of the signal. The Scalings page is there to give access to each message's input and output range. It only works with arguments that have a number value. It is displayed by choosing *View → Flip to Scalings page*, by pressing **⌘F** or by clicking on the tiny curved arrow button on the top right of the main window table.

One of the nice things with the Scalings page is that input and output ranges are scaled to [0 127] for MIDI signal, as seen for example for the first two values in the picture below. Note that the input min and max is not scaled because the input is an OSC message, presumably having an input range of [0.0 1.0]. Input range is described in further detail later in this section.

Message	In. min	In. max	Cap	Out. min	Out. max
/wii/1/accel/pry	0,0	1,0	<input type="checkbox"/>	0,0	1,0
0: pitch	0,0	1,0	<input type="checkbox"/>	0,0	1,0
1: roll	0,0	1,0	<input type="checkbox"/>	0,0	1,0
2: yaw	0,0	1,0	<input type="checkbox"/>	0,0	1,0
3: accel	-	-	<input type="checkbox"/>	-	-
/wii/1/button/A	0,0	1,0	<input checked="" type="checkbox"/> ②	0	127 ④
/wii/1/button/B	0,0	1,0	<input type="checkbox"/> ③	0,0	1,0
/wii/1/button/Down	0,0	1,0	<input type="checkbox"/>	0	127
/wii/1/button/Up	0,0	1,0	<input type="checkbox"/>	0	127

Description of the Scalings page columns:

1. The first three columns are the same as the one in the Routings Page (main window default view).

### 2. Input min - max

The input min - max range should describe the known range of a signal.

Typically [0.0 1.0] or [0 127] for MIDI signals.

### 3. Cap

When checked this checkbox activates a filter that excludes processing of the event when the input signal falls out of input range. For example if the input range is [0.0 1.0], cap is checked, and the input is 2.0, then the event will not fire. If the input signal is 0.5, the event will fire.

### 4. Output min - max

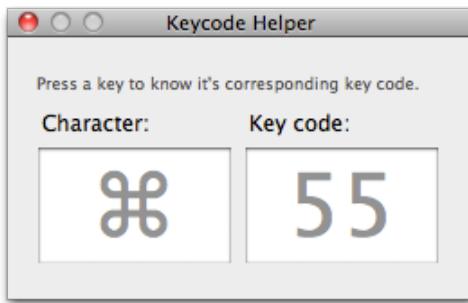
The output min - max range describes the desired range for an output signal.

Typically [0.0 1.0] or [0 127] for MIDI signals.

## Notepad

The Notepad is an embedded minimal rich text editor that is useful for taking notes or adding documentation. It is possible to change formatting, display a ruler, and use hyperlinks with the Format menu. It is also possible to paste or drag and drop images in the Notepad.

## Keycode helper

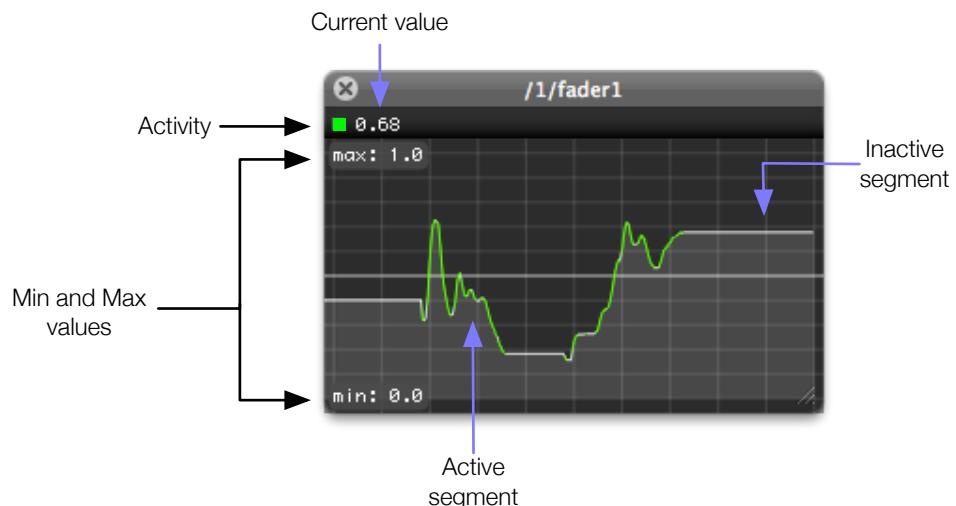


The Keycode Helper is a tool to find the Keycode corresponding to the press of a key. This tool is useful when using the Keycode Event type.

## Quick Look

The Quick Look window is displayed when the Space bar is pressed on a selected message. Currently, it only works with numerical values.

An activity monitor is green if signal is currently being received. The last received value is displayed next to it. The *max* and *min* indicators are just reminders of the Output Min and Output Max values of the message. The graph is scaled accordingly to this range.



It is useful to display the evolution of a signal over time. Each horizontal subdivision corresponds to one second of elapsed time. The windows shows a total of 10 seconds. Only green segments correspond to activity that has been recorded in the last instant. When the segment is drawn using a grey line, it means that no signal has been recorded.

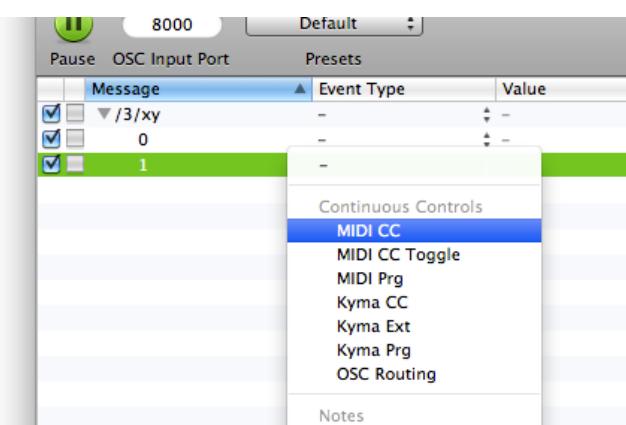
# Operations on messages

## Assigning an event

Once a message is received it is displayed in the Main Window as a new entry. The most elementary thing that can be done is to assign an event to the message or to one of its arguments.

The Event Types are discussed in more detail in the [Events](#) chapter.

To assign an event, first choose an Event Type by clicking on the drop down menu of the corresponding column:



Once the Event Type is selected, a Value can be chosen from the menu in the next column.

### Tip: Turbo assign events

It quickly becomes tedious to configure a lot of events if they all have the same Event Type. For example, it is common to configure many *MIDI Control Change* events with sequential numbers.

Most items of the main window can be used while multiple rows are selected.

In addition, when choosing an item from the Value menu, sequential numbers will be chosen if the Option ⌘ (aka Alt) key is held down while clicking on a Value.

Typical scenario: [Assign MIDI CC events to many messages, using sequential CC numbers](#)

- ▶ Select multiple messages
- ▶ Choose *MIDI CC* in the Event Type column
- ▶ While holding the Option ⌘ key, choose the item 0 in the Value column

## Edit menu

Most of the operations on a message are performed through the commands in the Edit menu.

### Copy

⌘C

### Paste

⌘V

### Cut

⌘X

Use the Copy and Paste functions to copy and paste messages and events between different presets or windows.

### Delete

Backspace Key

Delete one or more events by selecting them in the list and pressing the Backspace key.

Note that if you delete a message, it will not prevent OSCulator from receiving it again. Deleting a message is useful when you want to clean up the document, or if you want to reset a message to its initial state.

### Select All

⌘A

Select all messages.

### Deselect All

⇧ ⌘A

Clears the current selection.

### Duplicate

⌘D

Use the Duplicate command to send the same message to many events.

For example, with a single button of the Wiimote you can trigger two MIDI Control Change events. For this, select the message that corresponds to the button, and choose *Edit → Duplicate*. This will create two entries that you can respectively assign to two events. You can repeat that operation for as many copies as you want of the same message.

### Split

⇧ ⌘D

“Splits” an incoming message in two parts.

This function creates two duplicates with predefined scalings, one for the low values of the incoming message (between 0 and 0.5, excluded) and the other one for high values of the incoming message (between 0.5 and 1). The outputs are scaled to [0 1]. This is very useful when you have the input from the axis of a controller and want to assign a different event for each side of the axis. For example, with the joystick of the Wiimote's Nunchuk, you can assign a different event when the joystick is on the left, and another for the right.

### Demux

⌃D

Demuxes or “Demultiplexes” the input based on its value. This function creates a new message for each of the different values it has received. When called, the existing events will be removed, waiting for the next incoming input. The message marked for demultiplexing is coloured in purple.

A simple use of that feature is to assign a different event to the same Wiimote button (e.g. /wii/1/button/A) whether it is pressed or released. Buttons send two values:

1 when the button is pressed

0 when the button is released

Select the message /wii/1/button/A and choose *Edit → Demux*, then press and release the button again. The message then becomes :

```
/wii/1/button/A  
0      -  
1      -
```

Be careful not to use this feature with float numbers as it could produce nonsense results.

## View menu

### Parameters...

Display the [Parameters window](#). This is an essential part of the application where advanced parameters are specified, like *Key Combos* or *OSC Routings*.

### Show Wiimote Drawer

Display the Wiimote drawer.

### Flip to Scalings Page / Flip to Routings Page ⌘F

Toggles between Scalings and Routings page. For more information on the Scalings page, see the [Scalings page](#) section.

## Routing menu

### Start Routing / Pause Routing

This menu command has the same effect as the play button in the toolbar. When the routing is stopped, OSCulator will stop receiving OSC, MIDI, and all other inputs. Connected Wiimotes are not disconnected, but they stop sending their messages.

### Lock Input ⌘L

This function prevents the user from altering the state of the running document by adding messages to or removing them from the main list. This is useful when you are on a live show and want to keep the document from being modified accidentally.

### Solo ⌘R

Use this command to enable only the selected events. When this command is called again, it restores the previous state.

#### Ableton Live example:

A useful example of this feature is when you are mapping an X/Y pad in Ableton Live with MIDI Control Changes. When in learn mode, Live listens to all incoming MIDI messages. An X/Y touchpad in TouchOSC sends two MIDI messages (one for the X axis, and one for the Y axis), but Live cannot differentiate them, and will only map the selected control to the last received message.

To solve this, select the line you want to solo, hit ⌘R, and activate MIDI learn in Live. When you're done learning the MIDI event, go back to OSCulator and hit ⌘R again to restore the previous working state.

### Hide Toolbar

Hides the Toolbar, making the main window smaller.

### Customise Toolbar...

Presents a sheet window that allows customisation of the Toolbar.

### Enable ⌘R

Enable the selected messages.

### Disable ⌘R

Disable the selected messages (disabled messages do not trigger their events.)

### Invert output scaling ⌘I

Shortcut action that is the same as exchanging the output min and output max values in all the selected messages of the [Scalings page](#).

### Collapse / Expand ⌘E

Collapses or expands the selected lines.

### Quick Look

Pressing the space bar will display a small graph window useful to have a visual cue of the input data. See the [Quick Look](#) section.

## Manually Create Message...

⇧⌘M

Prompts the user with an OSC message name, and displays it in the list. This command is only useful in some precise situations:

One of them is with TouchOSC, when configuring bi-directional communications with a control that does not send messages, for example a label or a LED:

Choose *Routing* → *Manually Create Message...* and give the name of the LED you want to map in Live (e.g. /1/led1)

When the message is displayed in the list, assign a MIDI CC event

In Live, activate the MIDI learn mode (⌘M) and select the control you want to map with the LED

Back to OSCulator, click in the activity monitor cell of the /1/led1 message (this will have the effect of sending the assign MIDI message to Live)

Disengage learn mode in Live, and use the mapped control, this will create a back-mapping entry in OSCulator, thus changing the status of the LED in TouchOSC.

## Format menu

The Format menu is standard and is used with the [Notepad](#) window. It is used to customise the look of the text, change the justification, or insert hyperlinks.

## Window menu

### Keep windows on top ⌘T

Toggles between keeping OSCulator's windows on top of all others, or the normal behaviour.

#### Tip: Keeping the main window on top of all others

When you work with other software, it can become annoying to see other windows obscure OSCulator's window and [Quick Look](#) windows. In order to keep OSCulator's window on top of all others, like a floating tool, choose *Windows* → *Keep window on top*, or press ⌘T.

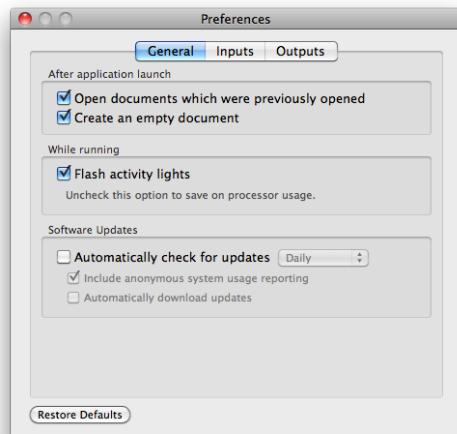


# Preferences

## General

### After application launch

- ▶ *Open documents which were previously opened*  
Check the box if you want to automatically open the documents that were left open when OSCulator has been quit.
- ▶ *Create an empty document*  
Check the box if you want to create an empty document each time OSCulator is launched.



### While running

- ▶ *Flash activity lights*  
The operation of flashing lights can be time consuming for the host processor. When you want to achieve the best precision (in terms of timing), and you own a computer with only one processor, this option can help you. It is better to keep this option checked if you own a multi-processor or multi-core computer, because they are not overly affected by the flashing operations.

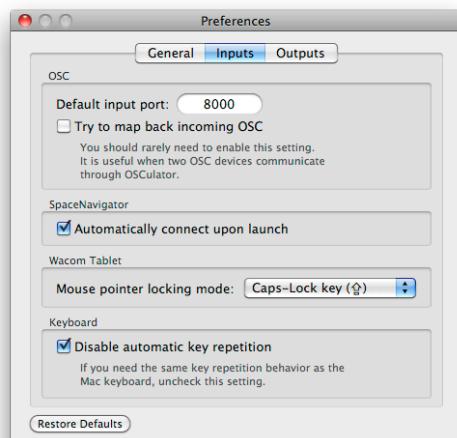
### Software Updates

Options to control the software updating process.

## Inputs

### OSC

- ▶ *Default OSC Port*  
OSC input port that will be used for new documents. The default value is 8000.  
Here is a list of useful ports:
  - The JazzMutant Lemur uses port 8000
  - SynthPond uses port 12345
  - reacTIVision or other TUIO enabled software use port 3333
  - Max/MSP often uses port 9000
  - lannix uses port 57120
- ▶ *Try to map back incoming OSC*  
Advanced feature used to automatically link two OSC devices through OSCulator.



### SpaceNavigator

- ▶ *Automatically connect upon launch*  
Every time a document is opened or created, OSCulator automatically uses the SpaceNavigator, unless it is already in use in another document.

## Wacom

- ▶ *Mouse pointer locking mode*

Choose the mode in which the Wacom tablet works, by locking the mouse pointer with the Caps-Lock key, a keyboard combo, or without mouse locking.

## Keyboard

- ▶ *Disable automatic key repetition*

Keyboard related events like Key Combo and Keycode support key repetition such as when a key is held on the keyboard. This option is disabled by default, meaning that holding the trigger on such event will result in only one key press.

## Outputs

### Mouse

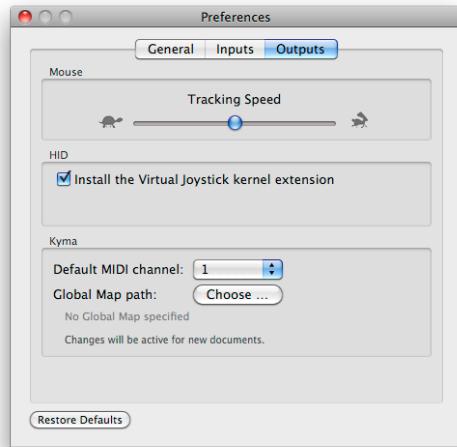
- ▶ *Tracking Speed*

By adjusting the slider, you can set the tracking speed of the Mouse (Relative Move) events. This works like the System Preferences Mouse tracking speed control.

### HID

- ▶ *Install the Virtual Joystick kernel extension*

The kernel extension is needed when the HID related events are used. Upon usage of one HID event, OSCulator will automatically ask if the extension should be installed. Disabling this option will uninstall the kernel extension.



### Kyma

- ▶ *Default MIDI Channel*

This is the MIDI channel that will be assigned to newly detected messages.

- ▶ *Global Map path*

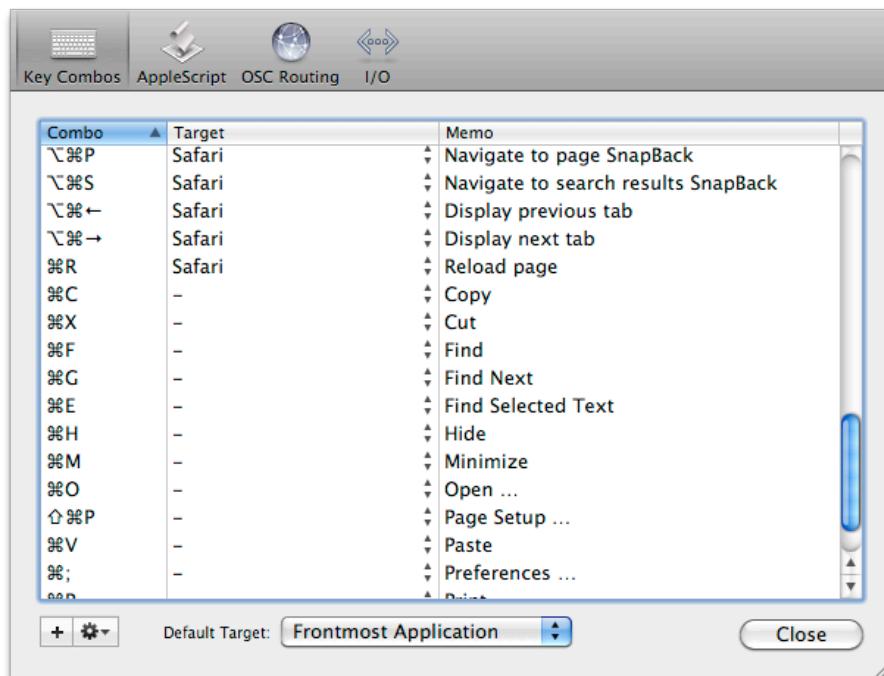
The Global Map is a clever way to automatically map incoming messages (e.g. control changes, note events, etc.) to Kyma Hot Values (e.g. **!Frequency**, **!Bypass**). Read more about Global Map in the Kyma manual. If you want to specify the Global Map you are using in Kyma, set its path here. Changes are taken into account the next time you create or open a document.



# Parameters window

The Parameters window can be accessed by clicking the Parameters button in the toolbar. This is where you can add new Key Combos, write Applescripts, define new OSC targets and routes, and change advanced parameters in the main configuration. This chapter covers the [Key Combos](#), [AppleScript](#) and [I/O](#) panels. The [OSC Routing](#) panel will be discussed in more detail in its own chapter.

## Key Combos



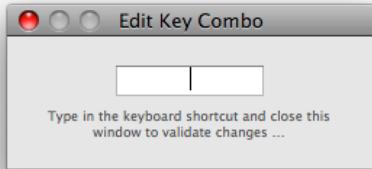
Before Key Combos can be used as events, they need to be defined in the Parameters window. Key Combos have the following properties:

- ▶ The key and its modifiers (Command ⌘, Option ⌘, etc.) that will be pressed when the event is triggered
- ▶ A target application: choose a running application from the drop down menu if you want to make an exception or choose the target application in the Default Target.
- ▶ A memo, to help remember its use.

## Define a Key Combo

To add a key combo, click the + button.

you will be prompted with a key combo edit window...



...then, type a key combo:



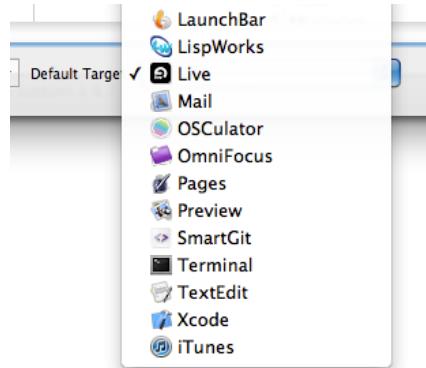
When you're done, close the window: the keyboard shortcut is now displayed in the list.

## Target applications

Because keyboard events (Key Combo, Keycode) are designed to work with applications running on the computer, it is possible to choose what target application these events should be sent to.

A default target can be defined to handle all keyboard events. When a default target is selected, all key combo and keycode events are exclusively redirected to that application.

If a target is specified but the application not present when the event is triggered, nothing will happen. On the other hand, if no default target is selected, any frontmost application will be used as a target.

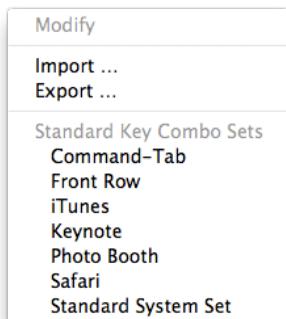


Key Combos offer an extra level of flexibility: for each key combo a specific target application can be defined, and takes precedence over the default target application.

## Import and Export Key Combo sets

A list of Key Combos can be loaded from or saved to a file. OSCulator comes with some predefined key combo sets, located in the Sample Patches folder.

When clicked, the gear button on the bottom left shows a menu, use *Import...* to load a key combo set, and *Export...* to save them to a file.



## Tip: How to find your Key Combos?

A document with many key combos can make it difficult to find the desired key combo in the Values list. One would wish to sort the Key Combo in one way or another. The Key Combo table in the Parameters window can be sorted by clicking on the column headers. This same sorting is applied to the Values menu list. For example, here is a Key Combo list sorted by the Memo key, and the associated Values menu:

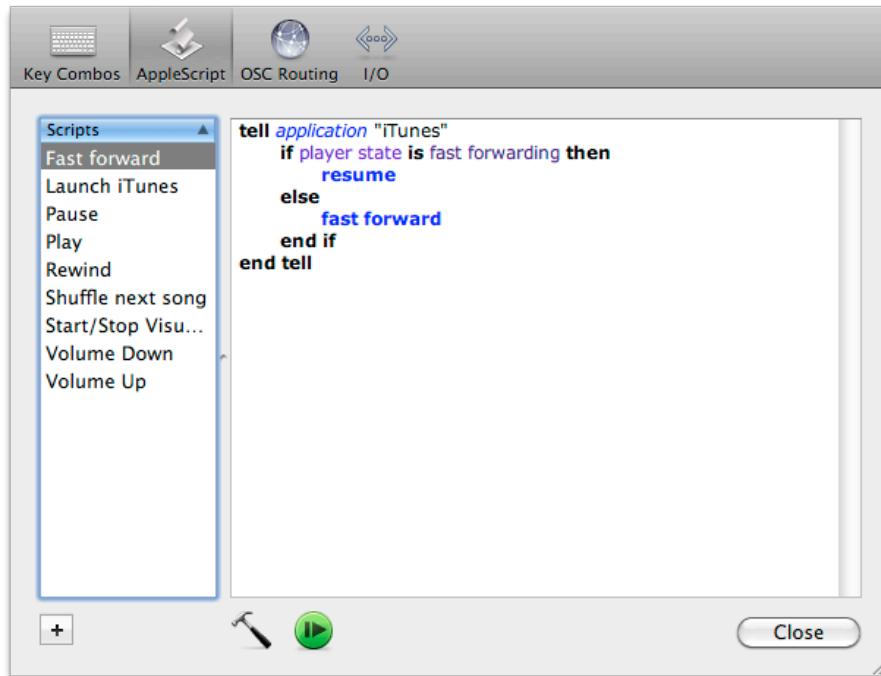
The 'Key Combos' window displays a table with columns: 'Combo', 'Target', and 'Memo'. The 'Memo' column contains arrows indicating the sorting order. The 'Values' menu window on the right lists various system key combinations, each preceded by a checkmark and an arrow indicating its current status.

Combo	Target	Memo
↶	-	▲ Close Front Row
⌘C	-	▲ Copy
⌘X	-	▲ Cut
⌘F	-	▲ Find
⌘G	-	▲ Find Next
⌘E	-	▲ Find Selected Text
⌘H	-	▲ Hide
⌘M	-	▲ Minimize
↓	-	▲ Navigate down
↑	-	▲ Navigate up
⌘O	-	▲ Open Front Row
⌘O	-	▲ Open ...
⇪ ⌘P	-	▲ Page Setup ...
⌘V	-	▲ Paste
↷	-	▲ Play and pause audio or video
⌘;	-	▲ Preferences ...

Values menu (sorted by Memo):

- Close Front Row
- Copy
- Cut
- Find
- Find Next
- Find Selected Text
- Hide
- Minimize
- Navigate down
- Navigate up
- Open Front Row
- Open ...
- Page Setup ...
- Paste
- Play and pause audio or video
- Preferences ...
- Print ...
- Return to a previous menu
- Select All
- Select an item in a menu or list
- Undo
- Volume down
- Volume up

## AppleScript



Like Key Combos, AppleScripts need to be defined first.

To create a script, click the '+' sign at the bottom left of the window. Double-click the script name to change it. Use the text editor to paste or write code.

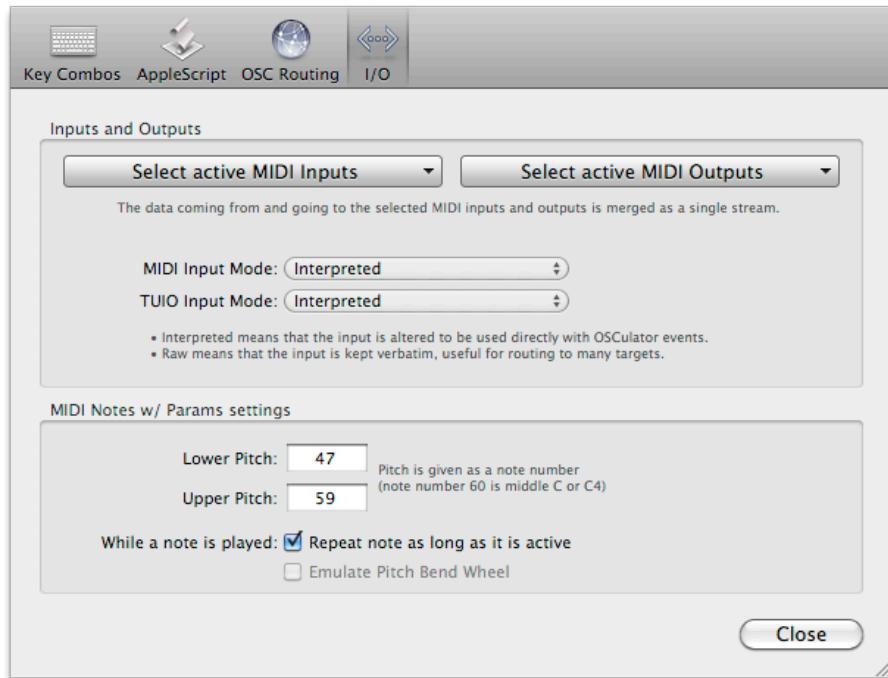
When clicked, the 'hammer button' will compile the script. If the script doesn't contain any error, the formatting of the script will be colorised to show it has been validated. You can also execute the script by clicking the green 'play' button.

As with other trigger based events, AppleScripts will be triggered when the received OSC value crosses 0.5.

**Tip:** It is a better practice to prototype and edit a script in the AppleScript Editor (found in the Applications folder).

This application gives better feedback on the evaluated results, and can display applications dictionaries.

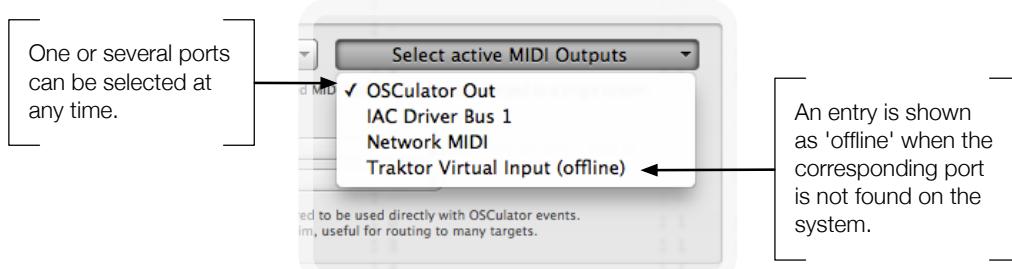
# I/O



The settings of this panel are related to MIDI with the exception of the TUIO input mode.

## Inputs and Outputs

This section enables the possibility to choose one or several input and output MIDI ports. These ports can be of any sort (Network MIDI, IAC Driver, physical MIDI interface ports).



Click on the button to display a menu with a choice of MIDI ports to choose from. When several MIDI inputs or outputs are selected, the streams are merged together. If it is needed to receive from or send to distinct MIDI port, the solution is to open another OSCulator window, and use different settings.

### Input modes

When OSCulator receives MIDI or TUIO data, it will ‘interpret’ the messages in order to display them in a meaningful way. For example, a MIDI Control Change #7 on channel 1 is received as /midi/cc7/1. It can be sometimes desired to remove that interpretation, for example when the raw MIDI stream must be sent to several destinations as an OSC message.

The input modes menu offer 4 choices:

Disabled	The function is not enabled.
Raw	The messages are the same as they were received.

<b>Interpreted</b>	The messages are translated into something easier to use with OSCulator events.
<b>Raw &amp; Interpreted</b>	The raw and interpreted messages are both displayed at the same time.

### MIDI and Kyma Notes w/ Params Settings

The MIDI Notes w/ Params (and its Kyma counterpart) events give a very flexible way to create MIDI Notes from any controller.

- Lower Pitch  
Sets the pitch note number associated with OSC value 0.0.
- Upper Pitch  
Sets the pitch note number associated with OSC value 1.0. Note: if Upper Pitch is lower than the Lower Pitch, then the note range is inverted.
- While a note is played: (does not apply to Kyma notes)
  - Repeat note as long as it is active  
As long as the OSC message holds the event, the MIDI note will be repeated in short intervals.
  - Emulate Pitch Bend Wheel  
Instead of repeating notes, this option produces a portamento-like effect by emulating the pitch bend wheel.



# Presets

## Concept

Presets are a powerful way to change the whole mapping configuration. If you are controlling a software, a fader sending a MIDI Control Change can be redefined to an OSC Routing.

This opens the door to a lot of creative opportunities, because it means you can use the same input for many different contexts, without having to manage multiple documents.



## Managing the presets

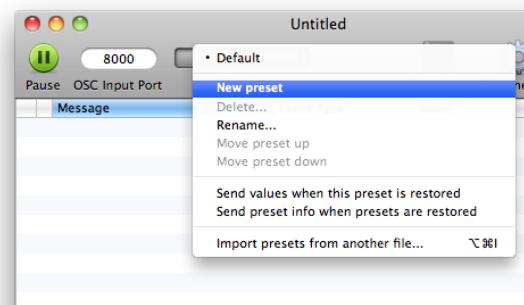
Presets are managed with the toolbar item that looks like a drop-down menu. If you don't have this item in your menu bar, select the *View* menu and choose *Customize Toolbar...*, then drag the *Presets Menu* onto the toolbar.

### Create a new preset

To create a new preset, click on the presets menu and choose *New preset*, this will copy the current preset to another one.

### Delete a preset

You can delete any preset as long as there is at least one left in the document. To do this, choose *Delete...* in the presets menu.



### Rename a preset

A preset can have any name. To change it, choose *Rename...* in the presets menu.

### Changing the order of presets

To change the position of the current preset within the list of presets, choose *Move preset up* or *Move preset down* in the presets menu.

## Preset change actions

### Send values when this preset is restored

This option applies to the current preset. Checking this item will make all messages in a preset send their values again.

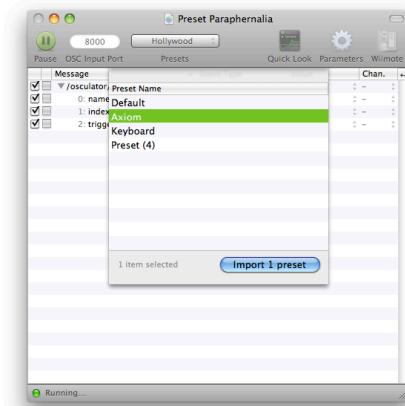
**Send preset info when presets are restored**

*This option applies to all presets.*

When a preset is restored, a `/oscillator/preset` message is triggered containing the **preset name** as an OSC string, the **preset index** as an integer number whose value goes from 0 to N, and a **trigger** argument that quickly goes from 1 to 0 that can be used for example to trigger a MIDI program change or a Key Combo event.

## Importing presets from other files

Pressing **⌘I** or selecting the corresponding menu option will present an open file dialog where the user can choose the file from which to import presets. Once the file has been chosen, a sheet window is presented with the list of presets. If no preset is selected, the operation can be canceled. The selected presets will be imported if the confirmation button is pressed.



# Events

This chapter lists the various types of events that can be associated with (i.e. mapped to) an OSC message, and describes in detail how each is used.

An event is the basic unit of work. It responds to its input by performing various actions. For example, send a MIDI Control Change or another OSC message, move the mouse, etc.

OSCulator's main window is used to connect an OSC message's value with an event. This is done by choosing an Event Type from the list of categories, and then choosing an Event Value. For events that need more advanced parameters, the [Parameters window](#) is used to provide additional configuration options.

An event processes one value at a time — these values are usually numbers or strings<sup>6</sup>.

## Overview

Most events are assigned to a message's argument(s), though those marked with an asterisk (\*) can be assigned to a message's address and/or argument(s).

*Click on an Event's name to go to the reference documentation.*

Control	
<a href="#">MIDI CC</a>	Converts the input value to a MIDI Control Change. The value is automatically scaled to the appropriate MIDI unit. An OSC value of 1.0 corresponds to a MIDI value of 127.
<a href="#">MIDI CC Toggle</a>	Similar to MIDI CC, except that the input is used as a toggle trigger, and the output can only be one of the two extreme values given by the output range (see Scalings).
<a href="#">MIDI Prg</a>	The input, used as a trigger, is converted to a MIDI Program Change when the input goes from a high value to a low value (for example, when a button which was pressed is released).
<a href="#">Kyma CC</a>	Similar to MIDI Control Changes, but with a much higher precision, and directly sent to the Kyma workstation.
<a href="#">Kyma Ext</a>	An extended set of Kyma CC including special events for sending Kyma compatible Pitch Bend, Channel Pressure, Wiimote and Tablet messages.
<a href="#">Kyma Prg</a>	Similar to MIDI Program Changes. Includes three special programs for selecting the Previous, or Next VCS preset, or for randomizing VCS values.
<a href="#">OSC Routing</a> *	One of the most versatile events, used to forward, construct and send OSC messages.

<sup>6</sup> there is an exception to this rule: the events OSC Routing and Console Log can also use an OSC message as input. That means that they will process the message along with all of its arguments.

Notes	
<b>MIDI Note</b>	Send a MIDI note with a fixed pitch. This event also gives access to Channel Velocity, Channel Pressure and Pitch Bend modifiers.
<b>MIDI Note w/ Params</b>	Advanced MIDI notes generation.
<b>Kyma Note</b>	Send a Kyma note with a fixed pitch. This event also gives access to Channel Velocity.
<b>Kyma Note w/ Params</b>	Advanced Kyma notes generation event. Unlike MIDI, the Kyma notes can be continuous (i.e. portamento effect).
<b>Note Params</b>	Stores the parameters used in MIDI Note w/ Params, and Kyma Note w/ Params.

Virtual Devices	
<b>Keycode</b>	Simulates a press on a specific key of the Mac keyboard. To be used in conjunction with the <a href="#">Keycode Helper</a> (⌘K).
<b>Key Combo</b>	Simulates a predefined keyboard shortcut.
<b>Mouse</b>	Moves the mouse pointer and simulates a press on a button or scroll wheel.
<b>HID</b>	Controls virtual joysticks exposed to HID compatible applications (most games for example).

Meta Events	
<b>Enable</b>	Controls the state of the Enable checkbox.
<b>Enable Toggle</b>	Like Enable, but treats the input as a toggle trigger.
<b>Enable Combine</b>	Like Enable, but also has the effect of triggering the targeted message.
<b>Presets</b>	A set of events that control the current preset.
<b>Change Channel</b>	Changes the channel of another message in the list based on the input value. A variant offers the possibility to change the channel of every message contained in a preset.
<b>Variable</b>	Writes the input in a variable, which can then be read with an OSC Routing.

System	
<b>AppleScript</b>	Triggers a predefined AppleScript.
<b>Console Log *</b>	Outputs the values of the message or the argument to the system's console.

Wiimote	
<b>Wiimote Enable</b>	Enable or disable a Wiimote's messaging with one event.
<b>Vibrate</b>	Turns on or off the rumble motor built-in a Wiimote.
<b>LEDs</b>	Changes the LEDs of a Wiimote.
<b>Motion Plus Reset</b>	Resets the absolute orientation of the computed attitude angles.



# MIDI output

MIDI messages are produced using related events. MIDI CC, MIDI Prg, MIDI Note are a few examples of them. When triggered, those events send MIDI messages to the active MIDI ports<sup>7</sup>.

## MIDI CC<sup>8</sup>

This event uses numerical values to send a MIDI control change message on the active MIDI outputs. In the main window interface, use the *Value* column to choose a Control Change number, and *Channel* menu to choose the MIDI channel.

The expected input range is :

- ▶ for float input: 0.0 to 1.0. The output is then automatically scaled to the full MIDI scale from 0 to 127
- ▶ for integer input: 0 to 127. No automatic scaling is done<sup>9</sup>
- ▶ Other scalings can be obtained by editing the [Scalings page](#)

**Tip:** When displaying the Values menu for MIDI CC, press the Control key to display the names of the General MIDI (GM) controllers.

**Tip:** Learn how to mass assign MIDI CC events by reading the “Turbo assign events” tip.

## MIDI CC Toggle

A variation of MIDI CC, except the input is filtered to act as a toggle switch<sup>10</sup>. Consequently, this event has only two output values. These extreme values can be customised in the [Scalings page](#) by changing the Output Min and Output Max fields.

## MIDI Prg<sup>11</sup>

This event is used to send MIDI program change and gives two different ways of using it:

- ▶ a fixed mode where a *MIDI Prg* is chosen with a number ;
- ▶ *Change by Value* where the program change number is given by the input value.

### Fixed program change number

In the main Window, use the *Value* column to choose a program change number.

---

<sup>7</sup> see [I/O parameters](#) for setting the active MIDI output ports.

<sup>8</sup> Control Change

<sup>9</sup> for detailed information on input range automatic scaling, see [Event input range](#) in the Appendix.

<sup>10</sup> for detailed information on how trigger detection is made, see the [Event triggering](#) appendix.

<sup>11</sup> Program Change

## Change by Value

*MIDI Prg* offers another mode of operation where the input value is used as the Program Change number. In the Value menu, choose *Change by Value*.

The expected input is an integer value between 0 and 127.

# MIDI Notes and related events

Two events can be used to produce MIDI notes:

1. *MIDI Note* is easy to use and fits most situation, but may lack flexibility ;
2. *MIDI Note w/ Params* is designed for more complex MIDI note generation.

## MIDI Note

To configure a MIDI Note event, select MIDI Note in the Event Type column, and the desired note name in the Value column.

### Tip: Select a note by note number

MIDI notes are either represented by their note name (A#3 for example) or their note number (70). The Event Value menu shows the note names by default, but the note numbers can be displayed as well if the Control key is held down.

The channel of the MIDI note is selected as usual using the menu in the last column of the main window.

OSCalculator defines **note 60 as C3** (Yamaha convention, adopted by Ableton Live and Apple Logic Studio).

The MIDI Note event filters its input so it can detect a trigger, which makes it compatible with most OSC messages that have a numerical argument. For example when assigning a MIDI Note to the message sent from a controller's push button, or an accelerometer of the Wiimote.

For precise information on how this event is triggered, please see the [Event triggering](#) appendix.

## Channel Velocity

Notes are produced with a velocity setting (127 by default) that is shared amongst all notes of a given channel. Use *Channel Velocity* to change this setting. Please note that this event does not send any MIDI message, it only alters the value of the shared velocity setting.

Expected values for *Channel Velocity* are:

- ▶ floating numbers from 0.0 to 1.0 corresponding to MIDI velocity from 0 to 127 ;
- ▶ integers from 0 to 127 in which case no scaling is applied.

## Channel Pressure

Unlike *Channel Velocity*, this event does send MIDI channel pressure messages. The expected values are the same.

## Pitch Bend

Use *Pitch Bend* to send pitch bend MIDI messages.

Expected values are:

- ▶ floating numbers from 0.0 to 1.0 corresponding to MIDI velocity from 0 to 16383 ;
- ▶ integers from 0 to 16383 in which case no scaling is applied.

## MIDI Note w/ Params

*MIDI Note w/ Params* gives a more flexible way of producing MIDI Notes from OSC messages. It allows to control the individual pitch, velocity and aftertouch of a single note, and to play multiple notes at once by using a voice index<sup>12</sup>.

Here is how it works:

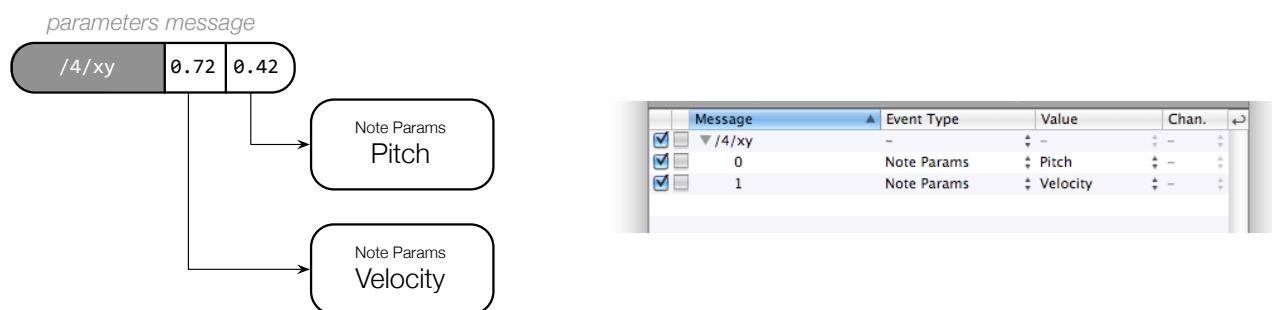
1. Information about the next note's parameters is stored ;
2. The note is built using the previously stored information and sent to the MIDI outputs.

It is best used with two OSC messages: one for the note parameters, and one for the note trigger.

Storing the note information is done using a *Note Params* events assigned to the parameters of an OSC message.

It does not have any effect other than remembering a named parameter belonging to the next played note. It works in the same way as *MIDI Note / Channel Velocity* except it is not acting globally.

### First step: Configure the note parameters

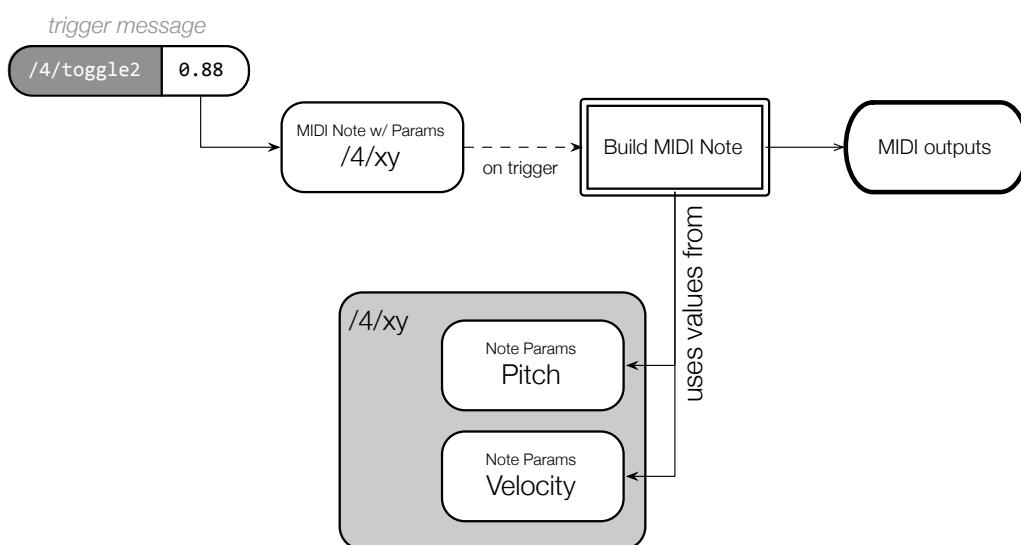


On the left, when the message */4/xy* is received, the parameters *Pitch* and *Velocity* are temporarily stored so that the values are made available for later use by a *MIDI Note w/ Params*. The picture on the right shows this same example in context.

### Second step: Assign the MIDI Note w/ Params

The event should be assigned to a OSC message that acts as the trigger of the note.

As shown in this picture, the OSC message */4/toggle2* is used to control the trigger of the *MIDI Note w/ Params* event. When the trigger is activated, it has the effect of building a note with the parameters previously recorded with *Note Params*.



<sup>12</sup> The voice index is a user-chosen integer number that is used to identify a note in a chord.

Here we show the final settings in the main window:

Pause	OSC Input Port	Presets	Quick Look	Parameters	Wiimote
	Message	Event Type	Value	Chan.	↔
<input checked="" type="checkbox"/>	/4/toggle2	MIDI Note w/ Params	↳ /4/xy	↳ 1	↳ ↳
<input checked="" type="checkbox"/>	↳ /4/xy	-	↳ -	↳ -	↳ ↳
<input checked="" type="checkbox"/>	0	Note Params	↳ Pitch	↳ -	↳ ↳
<input checked="" type="checkbox"/>	1	Note Params	↳ Velocity	↳ -	↳ ↳

### Expected values

Event	Expected input range	
	Float	Integer
<b>MIDI Note w/ Params</b>	<i>any value that activates the trigger</i>	
Note Params <b>Pitch</b>	[0.0 1.0], the value is scaled to match the MIDI pitch range as defined in the <a href="#">I/O Parameters</a> .	
Note Params <b>Velocity</b>	[0.0 1.0], then scaled to [0 127].	[0 127], unaffected by scaling.
Note Params <b>Aftertouch</b>		
Note Params <b>Voice</b>	<i>invalid</i>	<i>any positive integer</i>

### About the Voice Index parameter

The Voice parameter allows to control the note status (on or off) of several notes with a single OSC message. When the MIDI Note w/ Params event is triggered, the note status is saved with its corresponding voice index. This allows for example to turn on and off several notes for different pitches.

# OSC Routing

OSC *Routings* are like templates for OSC messages. With them, new OSC messages are built and sent to other devices.

Because of their flexibility, they are a little more complex than usual. It is recommended to read the first chapter to understand the terms used throughout this section.

## Overview

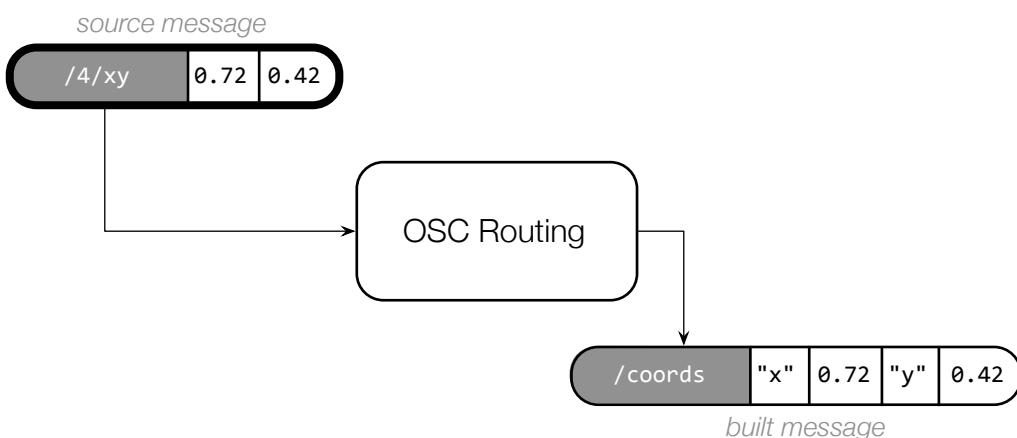
OSC *Routing* is one of the few that can be assigned to a message or a message's argument<sup>13</sup>. The distinction is important as it leads to different behaviours — it is also an often misunderstood feature.

### Routing a whole message

Message	Event Type	Value	Chan.
<input checked="" type="checkbox"/> <input type="checkbox"/> <b>/4/xy</b>	OSC Routing	↳ /coords : "x", arg[0], "y", arg[1]	–
<input checked="" type="checkbox"/> <input type="checkbox"/> 0	–	–	–
<input checked="" type="checkbox"/> <input type="checkbox"/> 1	–	–	–

An OSC Routing assigned to a message.

When an OSC *Routing* is assigned to a message, every argument is used as its input. Thus it is possible to change the order of the arguments or, for example, interleave them with other arguments as shown in the following example:



In this figure, a message named `/4/xy` with two float arguments is used as the source. The OSC *Routing* builds a message that has the address `/coords`, with four arguments:

1. the string “x”,
2. the first source argument,
3. the string “y”,
4. the second source argument.

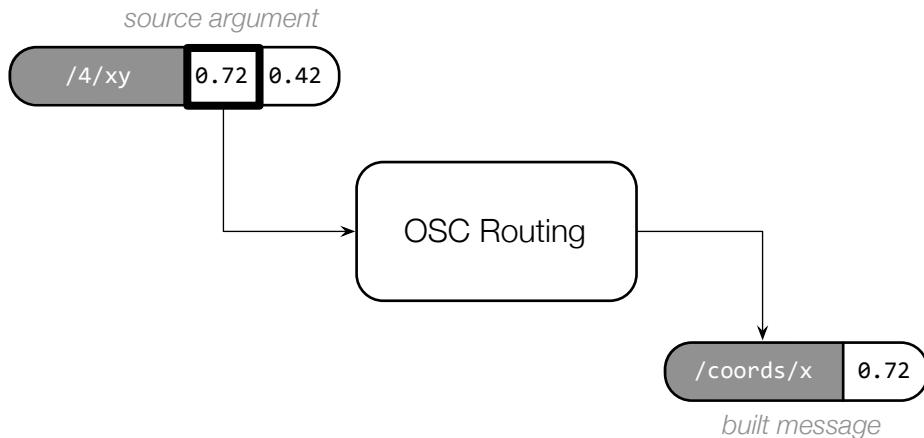
<sup>13</sup> see [The OSC message in detail](#).

## Routing a single argument

Message	Event Type	Value	Chan.
/4/xy	-	0.72	-
0	OSC Routing	D ➤ /coords/x	-
1	-	0.42	-

An OSC Routing assigned to a message's argument.

When an *OSC Routing* is assigned to a message's argument, then the only available information is the message's address and the argument's value.



Here, the *OSC Routing* is attached to the first argument of the source message.

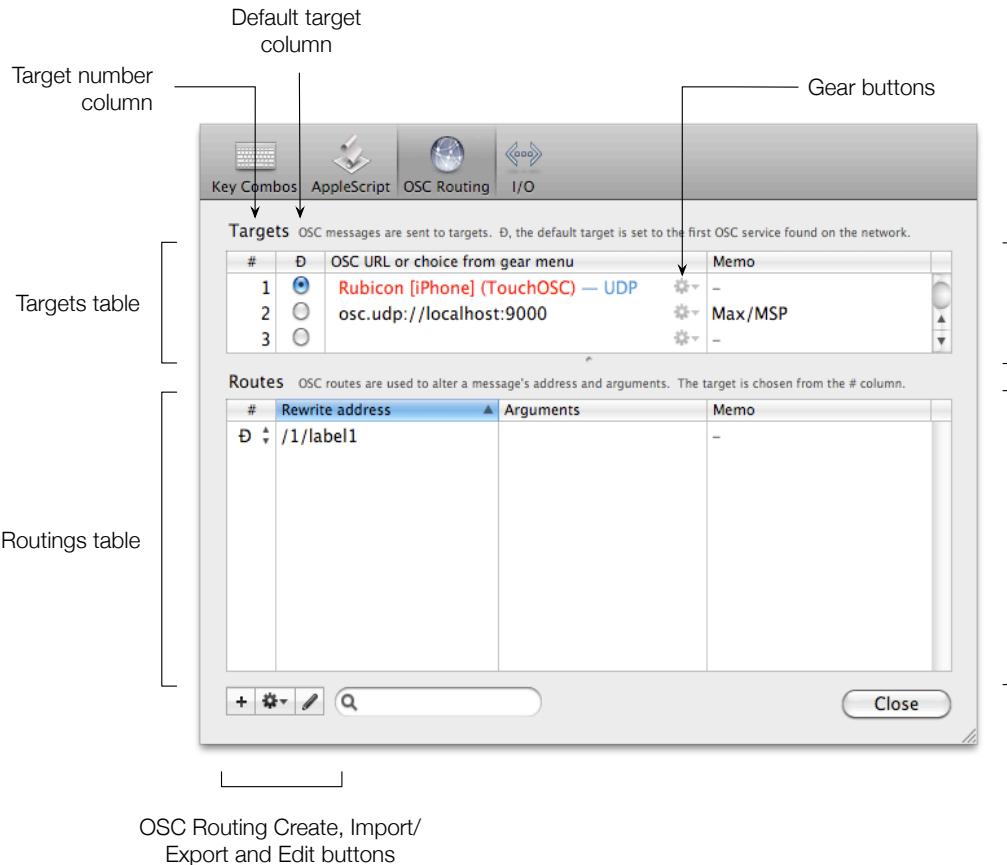
When no [rewrite address](#) is given, the address of the built message can differ from the source message's address. See appendix [OSC Address rewriting rules](#), for more information on the rules applied to single argument routing.

## When to use “whole message” vs “single argument” routing?

	Pros	Cons
<b>Whole message</b>	<ul style="list-style-type: none"> <li>• Full flexibility</li> <li>• Build complex messages</li> </ul>	<ul style="list-style-type: none"> <li>• May require more configuration work</li> <li>• Whole messages can not be Duplicated (see <a href="#">Duplicate</a>)</li> </ul>
<b>Single argument</b>	<ul style="list-style-type: none"> <li>• Easy to use</li> <li>• Can be Duplicated (see <a href="#">Duplicate</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• Resulting address is changed if no rewrite address is explicitly given</li> </ul>

# OSC Routing parameters

Configuration of OSC *Routings* is made in the [Parameters window](#), under the OSC Routing tab.



The configuration pane is divided into two parts:

▶ Targets table

This table lists all the external OSC servers with which OSCulator can send messages. Each of them is identified with a number. One server can be designated as the default target<sup>14</sup>.

▶ Routings table

This table lists all the user defined routings. Most of the time, clicking the + button to add one OSC *Routing* is enough to get started sending OSC messages. More advanced parameters will be described in the [OSC Routing Edit Window](#).

## Targets table

By definition, an OSC message must be sent to an OSC server, also named Target<sup>15</sup>.

The Targets table lists all known targets. There are three ways targets are defined:

1. Automatically discovered Bonjour services are added to the list. The first discovered service is incidentally made the default target.
2. Double-click in the host:port column and manually enter a *host:port* combination.
3. Click on the Gear button and select a Bonjour service or a predefined item.

<sup>14</sup> initially the first server discovered

<sup>15</sup> the OSC server does not need to be on another machine. A message can even be sent from OSCulator to itself.

## Routings table

Each routing in this table is also displayed in the main window, when choosing a value for the OSC *Routing* event type.

A routing is made of:

1. A rewrite address

This forces the OSC address of the resulting messages. By default, no value is provided, and the address of the source message is used.

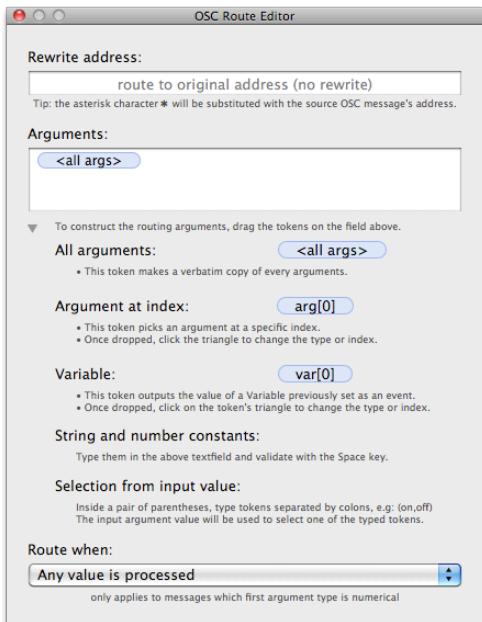
2. A collection of argument tokens

Arguments of resulting messages are defined by a collection of tokens. Tokens can be fixed values like a number (42) or a string ("violet"), or can be references to arguments of source messages.

When a new routing is created, it is configured to send messages to the default target. The default target is represented with the letter symbol **D**<sup>16</sup>. Other values are numbers corresponding to targets defined in the Targets table.

When a routing is double-clicked, the OSC Routing Editor Window is displayed. This is where the most specific configuration can be done.

## OSC Routing Edit Window



The OSC editor is divided into four parts:

1. [Rewrite address](#)

This is the address of the resulting message.

2. [Arguments](#)

This is the list of arguments that compose the resulting message.

3. [Tokens short memo guide](#)

4. [Trigger condition \(Route when\)](#)

Specifies the condition used to send the built message.

### Rewrite address

When no rewrite address is given, the resulting message's address is copied from its source message — in the case of single argument routing, some rules apply, see [OSC Address rewriting rules](#) in the Appendix..

If found in the rewrite address, the string <address> is substituted by the source message's address.

For example, suppose you need to prefix every message with the string /rosemary before sending them to another OSC device, the rewrite address should be: /rosemary<address>

Then, if this OSC Routing is used on a message named: /1/fader1,

the built message will have the address /rosemary/1/fader1.

<sup>16</sup> according to the [Wikipedia article](#): D — or D with stroke — was used in Medieval Latin to mark abbreviations of words containing the letter d. For example, h̄dum could stand for heredum "of the heirs". Similar strokes were added to other letters to form abbreviations.

## Arguments

This field lists the arguments of the built message by tokens. Tokens can refer to an argument of the source message, a variable, a string, a number, etc.

If there is no token in the argument field, the built OSC message will have no argument.

**<all args>** is a special token that copies every argument from the source message.

More options are accessible by clicking the triangle on a token's right corner. A menu is displayed with the token's index and the desired type.

With **arg[0]** (argument) tokens, the index is the position of the selected argument. Index 0 is the first argument of the source message, Index 1 is the second argument and so on.

With **var[0]** (variable) tokens, index is the number of the selected Variable (see Variable Event). Index 0 is the Variable 0, Index 1 is the Variable 1 and so on.

## Tokens short memo guide

This is a short reference guide on how to use available tokens. Template tokens are here to be dragged and dropped in the Arguments field.

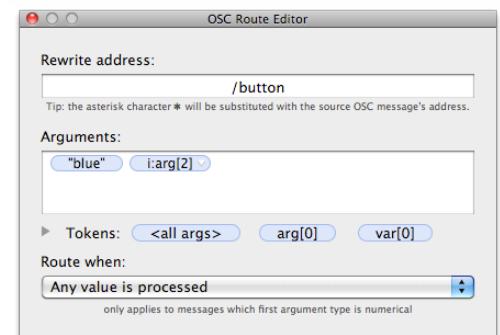
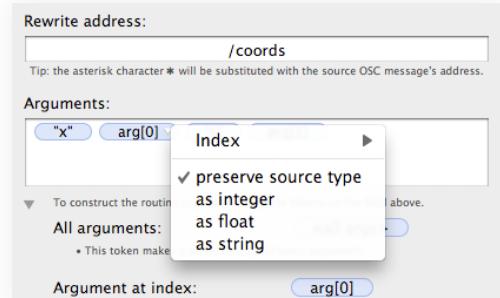
The memo text can be hidden by clicking the disclosure triangle.

## Trigger condition (Route when)

Provides a way to control when the message is sent. The condition only applies to source messages where the first argument is numerical.

The possible values are:

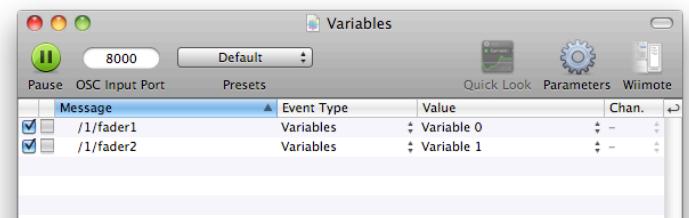
- ▶ The value crosses 0.5 upwards
- ▶ The value crosses 0.5 downwards
- ▶ The value crosses 0.5 in any direction
- ▶ The value goes from positive to 0
- ▶ The value reaches or leaves 0
- ▶ The value increases
- ▶ The value decreases



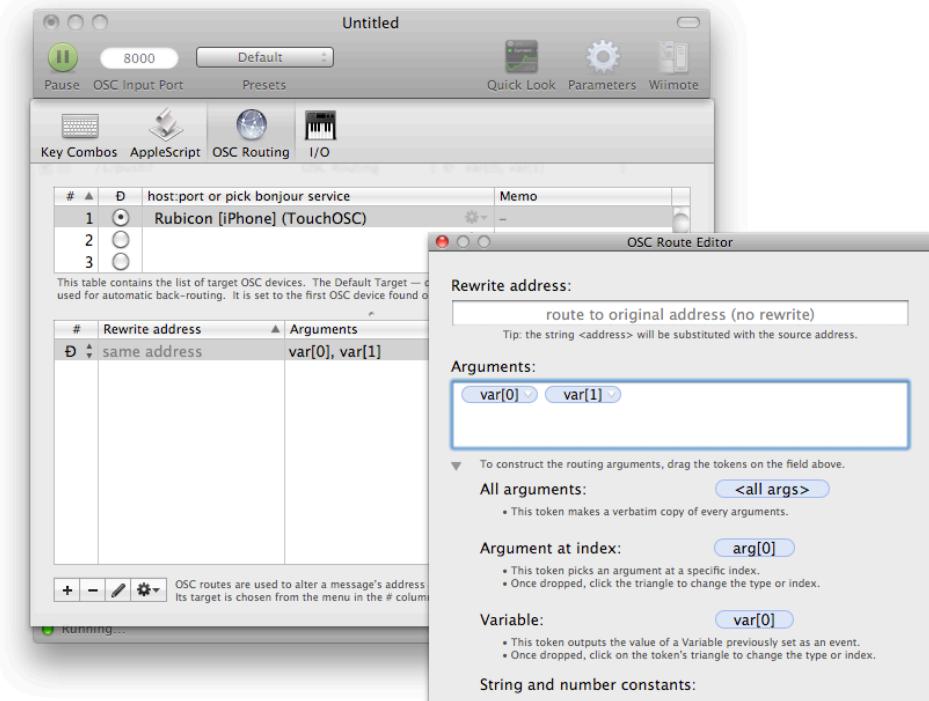
# Using Variables

Variables are used to store the last value it received and later use it in an OSC Routing.

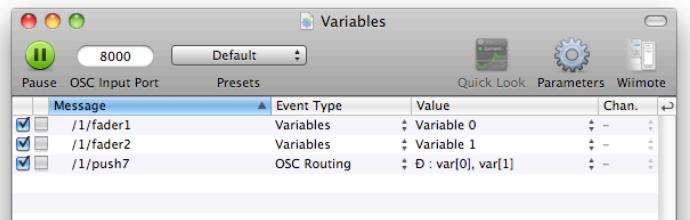
To store a value in a variable, first assign a Variable event to an argument. Here, two different variables are used to store the positions of two faders (TouchOSC is used as an example):



Then, an OSC Routing template is created using the two variables as template arguments. By consequence, the OSC messages generated from this template will have two arguments with the respective values of both variables.



Finally, the OSC Routing is assigned as the event of another message. Here, the message that triggers the OSC Routing is a push button:



# Virtual Devices

## Keycode

Simulates a press on a specific key of the Mac keyboard.

It is meant to be used in conjunction with the [Keycode Helper](#) ( $\text{⌘K}$ ) and has been designed to work seamlessly with most button messages produced by OSC controllers.

A keycode can be sent to the frontmost application or a specific target application<sup>17</sup>.

When several *Keycodes* are activated at the same time, they are combined together, just like on a regular keyboard.

The expected input for this event is fully described in the [Event triggering](#) appendix.

## Key Combo

*Key Combo* is used to send keyboard shortcuts to other applications. It is fully described in the [Key Combos](#) section.

## Mouse

*Mouse* is used to control the mouse pointer. There are several variants of this event, each of them controlling a specific feature of the pointer:

### Mouse / Relative Move X and Y

The mouse pointer is moved over the horizontal or vertical axis relatively to the current position. The amount of movement is given by the input that must be a float number between 0.0 and 1.0. When the input is 0.5, the pointer is not moved. The further the value is from 0.5, the further the pointer will move.

The acceleration effect can be adjusted in the Preferences in the Output tabs under the Tracking Speed setting.

### Mouse / Absolute Move X and Y

The mouse pointer is moved to a horizontal or vertical location in a space that is the union of all displays attached to the computer.

When the input is a float number, 0.0 on the X axis is the leftmost location, and 1.0 the rightmost location.

When the input is an integer number, 0 on the X axis is the leftmost location. The rightmost location is computed as the sum of every length on the X axis.

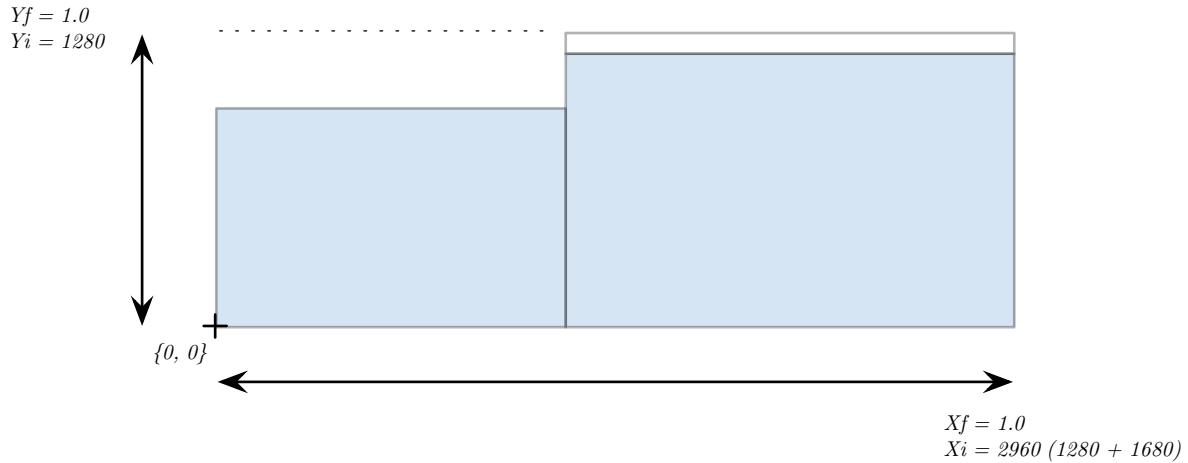
When the input is a float number, 0.0 on the Y axis is the bottommost location, and 1.0 the topmost location.

When the input is an integer number, 0 on the Y axis is the bottommost location. The topmost location is computed as the sum of every length on the Y axis.

---

<sup>17</sup> see [Key Combos](#)

The following diagram illustrates the *Mouse / Absolute Move* coordinate system with two monitors.



### Mouse / Trackpad Style

Provides the same relative movement behavior as a trackpad.

When using *Mouse / Absolute Move* with a standard XY mapped to your computer's pointer, touching the bottom left of the control would cause the pointer to jump to the bottom left of the screen, i.e. 'absolute' mapping. Which is not what happens when we use a *Mouse / Trackpad Style* to move the pointer.

For this event to work properly, three informations are needed:

- ▶ the current X coordinate (*Trackpad Move X*)
- ▶ the current Y coordinate (*Trackpad Move Y*)
- ▶ if the control is currently being touched (*Trackpad Touch*)

Note to TouchOSC users:

It is easy to assign the *Trackpad Move X* and *Y* events to each coordinate of a XY control. The touch parameter needs a little more work: To activate touch messages, open TouchOSC, go to Options and turn on "Send z messages".

You can now assign the *Trackpad Touch* event to the message ending with /z.

### Mouse / Buttons

Simulates the press or release of a mouse button. Left, Right and Center Button are also known respectively as button 1, 2 and 3 in other applications.

### Mouse / Scroll Wheel

*Scroll Up*, *Scroll Down*, *Scroll Left* and *Scroll Right* are events that move the view under the mouse pointer in the chosen direction when they are triggered.

*Smooth Scroll X* and *Smooth Scroll Y* will also move the view under the pointer in the X or Y axis but smoothly; like the scrolling on a Magic Mouse. The amount of movement is given by the input that must be a float number between 0.0 and 1.0. When the input is 0.5, the pointer is not moved. The further the input is from 0.5, the further the view under the pointer will move.

## HID<sup>18</sup>

Two devices are made available to compatible applications, namely “OSCulator HID 1” and “OSCulator HID 2”. These devices are not real HID gamepads, but virtual abstractions that are controlled from the values that are sent through OSCulator’s HID events.

The channel of the HID events is used to select which virtual device is controlled<sup>19</sup>.

Each gamepad is made of 2 analog joysticks, 16 buttons, and 4 “analog” buttons.

### Axes

The 4 events in this category are used to control the virtual joystick 1 and 2 on axis X and Y. Input is expected as float numbers between 0.0 and 1.0.

### Buttons

Each event is used to control a button. The input is used to trigger this button.

### Analog Buttons

Each event is used to control an “analog” button. The input is expected to be a float number between 0.0 and 1.0.



---

<sup>18</sup> see [Human Interface Device](#)

<sup>19</sup> there are by consequence only two channels (1 and 2) for HID events.

# Meta events

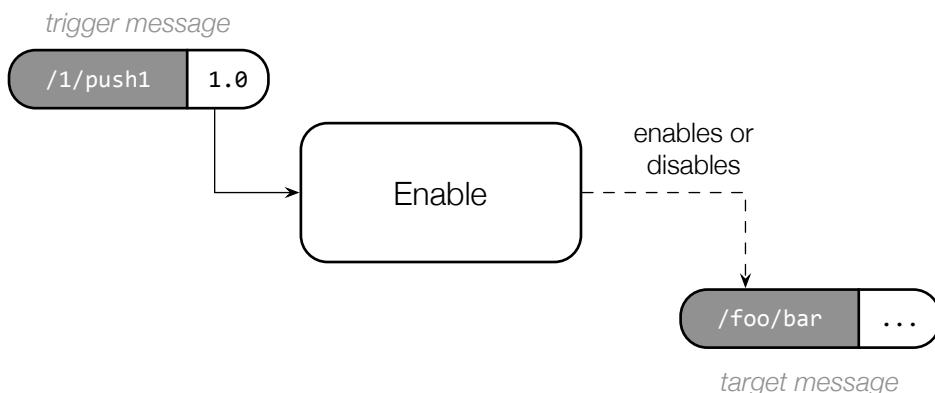
These events are used to change the state of the application itself. They are used to enable, disable or change the channel of another message, change the active preset, etc.

## Enable, Enable Toggle & Enable Combine

This family of events simply change the “enable” state of a message<sup>20</sup>. When a message is enabled, it processes its input which may in turn trigger the events. When a message is disabled, nothing is done.

Enabling and disabling messages can be useful for example when one want to send accelerometer data to a EQ filter only when a button is pressed.

All of those events work by assigning them to the message that is supposed to trigger the enabled state change, and choosing the Event Value which is the target message that the user wishes to enable or disable.



### Enable and Enable Toggle

The difference between these two events is how they are triggered. *Enable* simply enables its target when the input is above the trigger threshold, or disables its target when the input below the trigger threshold<sup>21</sup>.

*Enable Toggle* toggles between the enabled and disabled state. This is useful to use it for example with a controller button.

For more information on how events are triggered, please consult the [Event triggering](#) appendix.

**Tip:** In fact, the *Enable* family of events work by inverting the current enabled state of their target. If the target was initially enabled, it becomes disabled, and vice versa.

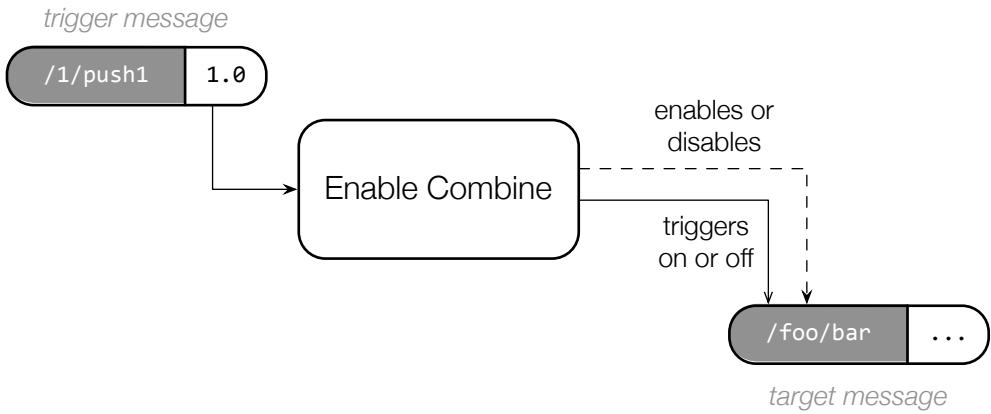
Be careful to check the right initial enabled state to get the desired results.

### Enable Combine

*Enable Combine* is a little bit more complex than its predecessor. It behaves like *Enable*, but it also forces the target message to trigger on when it is enabled, and respectively off when it is disabled. This is useful when messages that control MIDI Notes must be disabled. If *Enable* was merely used, MIDI notes that were on, would not be turned off.

<sup>20</sup> which can also be changed by clicking the “enable” checkbox located in the first column of the main window.

<sup>21</sup> the trigger threshold is defined as the middle point between the input min and input max, see [Scalings page](#).



Checkout the *Wii Guitar Example* file in the *Samples Library* folder for an example on how to use the Enable Combine event. In this example, the strum bar is used to play notes, and the colour key on the neck is used to select the notes. MIDI Note are produced only when both the strum bar and a colour key are activated together.

## Presets

[Presets](#) concepts are fully described in their own chapter. We will focus here on the events used to control activation of the presets. There are three ways to select a preset:

1. Use the events *Next Preset* or *Previous Preset*  
When triggered, these events change the active preset to the next or previous one.
2. Change by Index  
Sending an integer value changes the active preset to the one selected. The input value 0 corresponds to the first preset. Float numbers are allowed as they are automatically converted to integers without scaling.
3. Switch to a named preset  
When triggered this event changes the active preset to the chosen preset.

## Change Channel

This event allows to change the channel of all other messages in the current preset, or a specific message.

The expected input value can be:

1. a float in the range [0.0 1.0], then internally scaled so it corresponds to channels [1 16] ;
2. an integer in the range [0 15], corresponding to channels [1 16].

## Variables

Variables are described in the [OSC Routing](#) chapter. This meta-event is used to store any kind of value in a variable.



# MIDI input

For each opened document, OSCulator publishes a virtual MIDI port named “OSCulator Input (nnnn)”, where *nnnn* is the OSC input port of the document.

The intent of the MIDI Input is not to be exhaustive, but to provide a simple access to most useful MIDI commands.

## Received messages

	<b>Message Definition</b>	<b>Description</b>
<b>Note</b>	/midi/note/<channel>	Received whenever a MIDI note occurs. 3 arguments: the pitch of the note (ranges from 0 to 127), its velocity (0.0 to 1.0) and a value representing the state of the note (1 when the note is on, 0 when it is off.) For details on accessing the individual notes, please read the next section of this chapter.
<b>Program Change</b>	/midi/program/<channel>	Received when a program change occurred. 1 argument: the value of the program change (1 to 128.)
<b>Aftertouch</b>	/midi/aftertouch/<channel>	Received when the aftertouch changed for a given pitch and channel. 2 arguments: the pitch (from 0 to 127) and the aftertouch (0.0 to 1.0.)
<b>Channel Pressure</b>	/midi/pressure/<channel>	Received when the channel pressure has changed. 1 argument: the pressure (0.0 to 1.0.)
<b>Pitch Wheel</b>	/midi/wheel/<channel>	Received when the pitch wheel moved. 1 argument: the position of the wheel (0.0 to 1.0.)
<b>Control Change</b>	/midi/cc<number>/<channel>	Received when a control changed. 1 argument: the control value (0.0 to 1.0.)
<b>Clock Start</b>	/midi/clock/start	Received when the MIDI clock is started. <i>No argument.</i>
<b>Clock Stop</b>	/midi/clock/stop	Received when the MIDI clock is stopped. <i>No argument.</i>
<b>Clock Continue</b>	/midi/clock/continue	Received when the MIDI clock continues. <i>No argument.</i>
<b>Clock Position</b>	/midi/clock/position	Received when a MIDI clock position event occurred. 2 arguments: the beat as an integer from 0 to 3, and the sub-beat (or tick) and an integer from 0 to 5.

## How to use the /midi/note/<channel> message in practice?

The /midi/note message is a bit tricky because the pitch is expressed not in the address, but as an argument. This means that other arguments values are dependent to the pitch argument value.

	Message	Event Type	Value	Chan.	↔
<input checked="" type="checkbox"/>	/midi/note/1	-	▲ -	▼ -	▲ ▼
<input checked="" type="checkbox"/>	0: pitch	-	▼ -	▲ -	▲ ▼
<input checked="" type="checkbox"/>	1: velocity	-	▲ -	▼ -	▲ ▼
<input checked="" type="checkbox"/>	2: trigger	-	▼ -	▲ -	▲ ▼

It is more interesting to trigger events based on the pitch of a note, therefore we must find a way to display the velocity and trigger in terms of the pitch, like a drill down navigation.

This filtering process is called *demultiplexing* or in short *demux*.

To demultiplex a /midi/note/<channel> message in function of its pitch, first select the pitch argument, and then choose *Edit → Demux* (or ^D). The message will turn to purple and its arguments will disappear, waiting for new information.

	Message	Event Type	Value	Chan.	↔
<input checked="" type="checkbox"/>	/midi/note/1	-	▼ -	▼ -	▲ ▼

At this moment, you must send new MIDI notes in order to have the individual pitches appear:

	Message	Event Type	Value	Chan.	↔
<input checked="" type="checkbox"/>	/midi/note/1	-	▼ -	▲ -	▲ ▼
<input checked="" type="checkbox"/>	▼ 48	-	▼ -	▲ -	▲ ▼
<input checked="" type="checkbox"/>	0: velocity	-	▲ -	▼ -	▲ ▼
<input checked="" type="checkbox"/>	1: trigger	-	▼ -	▲ -	▲ ▼
<input checked="" type="checkbox"/>	▼ 53	-	▼ -	▲ -	▲ ▼
<input checked="" type="checkbox"/>	0: velocity	-	▲ -	▼ -	▲ ▼
<input checked="" type="checkbox"/>	1: trigger	-	▼ -	▲ -	▲ ▼
<input checked="" type="checkbox"/>	▼ 60	-	▼ -	▼ -	▲ ▼
<input checked="" type="checkbox"/>	0: velocity	-	▲ -	▼ -	▲ ▼
<input checked="" type="checkbox"/>	1: trigger	-	▼ -	▲ -	▲ ▼

It can be noted that this time, the pitch is displayed as a new sub-message, and the demultiplexed argument has been removed. Now, each velocity and trigger arguments are attached to a corresponding pitch.

## Automatic back-mapping in practice:

### Ableton Live configuration

When a MIDI message is received for the first time, OSCulator searches for a corresponding OSC message that is at its origin.

For example, suppose you have a fader in TouchOSC that sends the OSC message /1/fader1 and is configured in OSCulator as a MIDI CC 7 on channel 1. When OSCulator receives the MIDI message /midi/cc7/1, it will search for an OSC message that was at its origin, and find the message /1/fader1.

If OSCulator finds such message, it will create an “inverse” event (in the example case, this will be an OSC Routing event that routes back the MIDI value to the TouchOSC fader). This process is known as *back-mapping*.

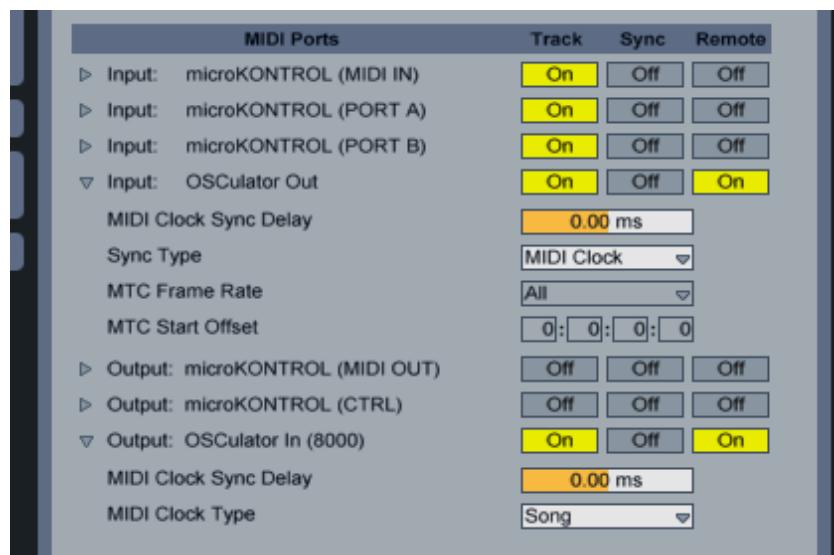
## Ableton Live configuration tutorial

This tutorial explains how to control a MIDI enabled software like Ableton Live from your iPhone.

### Ableton Live configuration

First of all, launch OSCulator and Live<sup>22</sup>.

In Live, open the Preferences, go to the *MIDI Sync* tab and enable the *Track* and *Remote* columns for “OSCulator Out” and “OSCulator In” MIDI ports. This enables Live and OSCulator to communicate through MIDI.



Ableton Live MIDI Sync Preferences panel

### TouchOSC configuration

Now, launch TouchOSC on your iPhone, iPod or iPad.

Please make sure you have the latest version installed.

- ▶ Under connections, tap on the OSC section. You should see a “Found Host” with your computer’s name and mentioning OSCulator, tap on this item. TouchOSC will automatically fill the fields required for the network communications.
- ▶ In the “Port (incoming)” field, enter 9000. This can be actually any port number you wish. 9000 is cool.
- ▶ From there your OSC settings should look like this (the “Host” field will be different though.)
- ▶ Tap on the back arrow, and select a layout, for example *Beatmachine*.
- ▶ Tap on the blue *Done* button and leave TouchOSC running.



### OSCulator automatic configuration

From there, we can either load a template or configure OSCulator manually. If you prefer to “plug and play” with MIDI, have a look at the Samples Library that you had with your installation of OSCulator, and just use the pre-canned template corresponding to the layout you chose (*Beatmachine* in our case).

You can also learn how to make such a template by following the next section.

<sup>22</sup> the order in which they are launched does not matter.

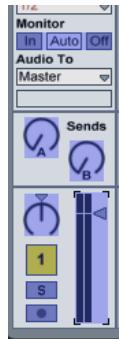
## OSCulator manual configuration

We will configure the yellow fader so it controls the volume of the first track in Live:

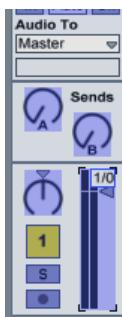
- ▶ In TouchOSC, touch the yellow slider to send an initial event.
- ▶ In OSCulator, the event `/1/fader1` should appear in the main window.
- ▶ Select “MIDI CC” as the Event Type, and “0” as the Value (any number will do in fact.)

Now, let's tell Live we want to use this MIDI control change to control the volume fader of the first track. This procedure also works for any button or control in Live.

- ▶ Return to Live, and click on the MIDI button, located upper right (or press  $\text{⌘M}$ ). Live turns to MIDI Learn mode.
- ▶ Click on the volume fader of the first track. The track should now look like this:



In TouchOSC, touch the yellow slider again. This will forward the event to Live, which will learn that we want to use the MIDI control change 0 to control the volume fader. A “1/0” label is now displayed next to the volume fader, like this:



- ▶ Leave MIDI learn mode by clicking on the MIDI button upper right (or press  $\text{⌘M}$ ).
- ▶ You're done!

If you click the volume fader in Live, you will notice OSCulator will automatically detect a MIDI message and will convert it to an OSC message that it sends to TouchOSC running on your iPhone.

That means that TouchOSC and Live are now synchronised both ways.



# Wiimote

## Support

OSCulator supports genuine Nintendo® devices: the Wiimote, Wiimote Plus and the Balance Board. Third party devices and extensions can work and most of them do, but it is not guaranteed that all are perfectly compatible.

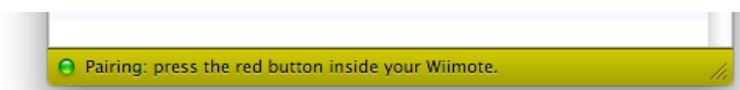
Up to 4 Wiimotes can be reliably connected to OSCulator. Some users reported that they have been able to use up to 7 Wiimotes on unibody MacBook Pros.

## Connecting the Wiimote

Before a Wiimote can be used with OSCulator, it must first be paired with your computer. This operation is done once per device. After that, the Wiimote can be connected simply by pushing a button.

To pair a Wiimote:

- Open the Wiimote drawer: choose *View → Show Wiimote drawer*<sup>23</sup>, and click on the button *Start Pairing*. This will scan the computer's neighbourhood for Wiimotes that are accepting connections.



- Open the Wiimote battery compartment, and press the **red button**.

Once messages begin to flow in the main window, you'll know the Wiimote is connected.

A green checkmark icon is also displayed in the Wiimote drawer, and the OS X bluetooth system menu shows that a *Nintendo* device is connected.

Once a Wiimote is paired with the computer, there is no need to repeat the pairing process. You can verify that your device is properly paired by looking at the device in the Bluetooth Preferences pane:



<sup>23</sup> alternatively, click on the Wiimote icon in the toolbar to show the drawer.

In the Bluetooth Preferences Pane, the *Paired* field is set to **Yes**.

## Troubleshooting

A Wiimote that is paired with your computer automatically opens a connection. That means that it can not be connected using the old way of pressing buttons 1&2 together. This can be a problem if you want to use a software that does not support the new pairing method.

It could also happen that the pairing information becomes invalid: the remote fails to store pairing information and accept to connect to the computer only when pressing the red button. An evidence of this problem is shown in the Bluetooth Preferences Pane in which the *Paired* field is set to **No**.

 In both case pairing must be reset.

To reset the pairing information:

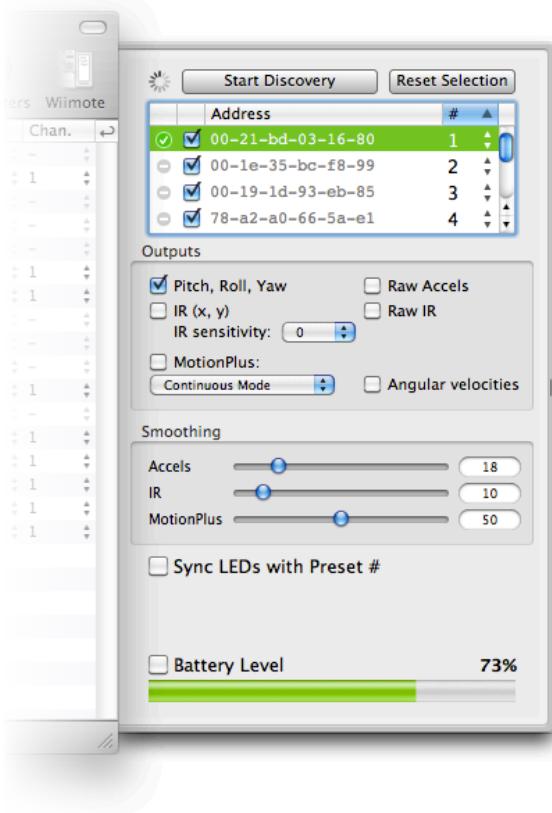
1. Open the Bluetooth Preferences Pane.
2. Make sure the Wiimote is disconnected from your computer. If the Wiimote is still connected, click on the bluetooth icon in the menu bar and select your Wiimote, and choose *Disconnect*.
3. Back to the Bluetooth Preferences, select the record corresponding to your Wiimote, and press the minus button to remove it.
4. Open OSCulator. *Hold the Alt (aka Option ⌥) key to start the pairing process in reset mode.*
5. Finally, press the red button on the Wiimote you wish to reset and wait for the process to complete.

## Connecting an extension

OSCulator will automatically detect and use extensions that are plugged on the Wiimote's extension port.

The MotionPlus device is an exception and needs to be enabled manually. To do this, check the MotionPlus button in the Wiimote drawer and wait for the calibration process to start.

# Configuring the Wiimote



## Wiimote drawer

The Wiimote drawer is used to connect new Wiimotes, organise them, and change their settings.

## Wiimote table

This table lists discovered devices:

- The first column shows an icon showing the device **status**.
- The next column is a checkbox that specifies whether a device is **enabled**.
- The third column is the **serial-number** of the Wiimote.
- The last column is the Wiimote's **slot**.

Entries in this table can be reverted to the factory defaults by clicking on the “Reset Selection” button.

## Wiimote slot

Each Wiimote has a slot number, which helps identifying one Wiimote from another. Another benefit of using a slot number is that a Wiimote can be swapped with another without affecting the configuration made in the main window.

The slot number is changed by choosing a new identifier in the drop down menu displayed right to the Wiimote address field.

## Settings

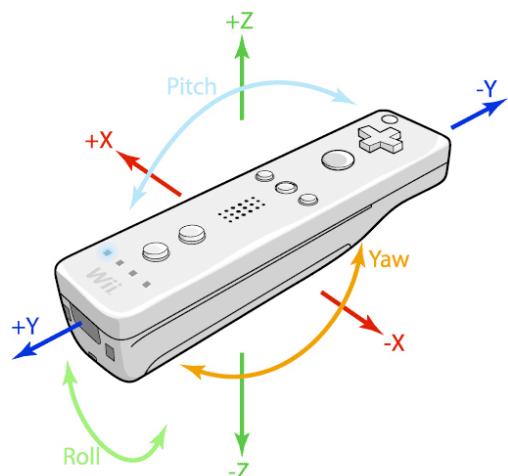
The way the Wiimote outputs data can be changed by the use of settings. These settings are shown in the Wiimote drawer and apply to the currently selected Wiimote. If multiple Wiimotes are selected, changes to the settings affect all selected Wiimotes.

## Outputs

In addition to triggers from the self-explanatory buttons on its surfaces, the Wiimote (and the Nunchuk) transmit measurements made by three accelerometers, each oriented in a different direction<sup>24</sup>.

The Outputs section is used to control how the Wiimote and extensions outputs data.

The settings in the left column are more related to data processed in a way that is directly useful, whilst the settings in the right column are related to “raw” data meaning that there is little or no processing after the measurement.



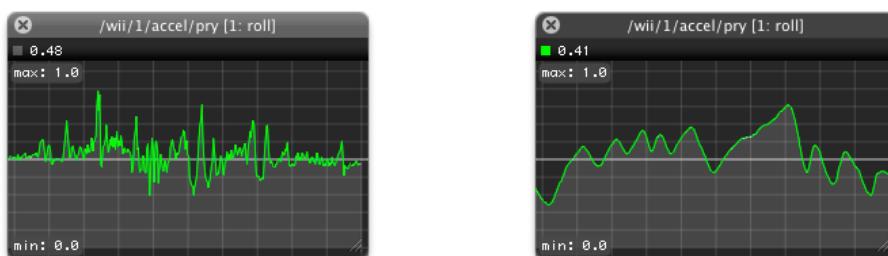
<sup>24</sup> for the story behind the development of this low-cost three-directional accelerometer chip, see [the story behind the development of the chip](#)

<b>Acceleration sensors</b>	Pitch, Roll, Yaw	These values are also called “attitude angles” and give an estimation of the orientation of the Wiimote. A fourth value “accel” is provided as the overall acceleration in all directions.
	Raw Accels (X, Y, Z)	Raw values as reported from the accelerometers, with some slight processing: the values are calibrated in the [0 1] range, and have smoothing applied (controlled by the Accels smoothing setting)
<b>Infrared sensor (IR)</b>	IR (X, Y)	When used in conjunction with the IR LEDs bar, bright spots of infrared light, or even two candles, this activates the readout of a virtual point in the space as X / Y coordinates.
	Raw IR	Reports information about 4 tracked bright dots as 4 groups of X / Y coordinates, with rough blob size and presence. Smoothing is applied (controlled by the IR smoothing setting)
	IR sensitivity	The sensitivity of the infrared camera can be adjusted with this setting. 0 is the default value, -2 is the less sensitive setting, and +2 is the most sensitive, meaning that dimmer objects will be better detected, but also unwanted noise.
<b>Gyroscopic sensors (MotionPlus)</b>	MotionPlus checkbox	This checkbox activates the search for a MotionPlus device, be it attached as an extension device or built in the remote (in some newer models). When activated, the MotionPlus always reports its values as Pitch, Roll, Yaw.
	Angular velocities	Raw values as reported from the gyroscopes on the device. An angular velocity is the speed at which a rotation is made. Using the angular velocity around the Y axis can be useful for example to build a “vinyl scratch” control.
	MotionPlus mode menu	This menu controls how the MotionPlus signal describes a full rotation of the Wiimote around each axis.

The messages produced by each activated setting is discussed further in this chapter.

### Smoothing

The Wiimote sensors produces a signal that can sometimes be too reactive. In order to smooth-out the response of those sensors, a smoothing setting is provided for each different part (Accelerometers, IR and MotionPlus).



*Effects of smoothing — on the left a value of 0 is chosen, on the right the value is 90.*

When the smoothing has a value of 0, no smoothing is applied, which means that the signal given is the same as the one measured by the device. In the case of the MotionPlus, a smoothing set to 0 gives more importance to accelerometers, while a value of 100 give more importance to gyroscopes.

The ‘Accels’ smoothing setting also applies to the Nunchuk, meaning that the ‘Pitch, Roll, Yaw’, and ‘Raw Accels’ outputs will be smoothed as well. There is no way to set a different smoothing for distinct output components.

### Sync LEDs with Preset #

When enabled, the current Preset number on the LEDs of the Wiimote.

When disabled, the LEDs show the Wiimote's slot.

### Battery Level

When enabled, the battery level is sent as a regular message, allowing to make it available to other applications, like Max/MSP.

## OSC messages

When a Wiimote is connected, messages are sent to OSCulator and displayed like any other messages in the main window. These messages can be configured by following the usual process. For example to send a MIDI control change that is based on the Wiimote acceleration, locate the message that is labelled /wii/1/pry/accel, select *MIDI CC* in the Event Type column, and choose any number you like from the Value column.

The Wiimote message addresses are decomposed as follows:

```
/wii/<id>{ /<extension>} /<control-type>{ /<acceleration-type>}
```

Address	Description
/wii	first, we have the /wii prefix.
/<id>	next, the slot number, helps you to differentiate each Wiimote
{ /<extension>}	optionally, if an extension device is connected, its name, which can be nunchuk or classic for the classic controller
/<control-type>	the control-type, which can be button, accel, ir, or joy (for joysticks)

Arguments of each kind of control type:

Control Type	Description	Arguments
.../button/<button-type>	Button	1 argument, the status of the button, 0 or 1.
.../accel/xyz	Linear acceleration	3 arguments: X, Y and Z accelerations. At 0g (no gravity), the value is centered on 0.5. The scale of 1g (4.8 m.s <sup>-2</sup> ) is 0.1. A readout of 0.6 means an acceleration of 1g, and 0.4, means an acceleration of -1g.

Control Type	Description	Arguments
<code>.../accel/pry</code>	Attitude angles	4 arguments, that correspond to the three attitude angles — Pitch, Roll and Yaw — and the scalar acceleration (combination of all the accelerations, which gives a meaning of the strength of the movement applied to the Wiimote.)  Attitude angles are centered (0 radian) when the value is 0.5. The scalar acceleration has the same units as the <code>accel/xyz</code> accelerations.
<code>.../joy</code>	Joystick	2 arguments: X and Y coordinates of the joystick.
<code>.../ir</code>	Infrared X/Y	2 arguments: X and Y coordinates.
<code>.../ir/xys/ [dot-id]</code>	Raw Infrared	if you have enabled Raw IR output, you can track up to 4 IR bright dots. The suffix <code>xys</code> is added with the id of the tracked dot. 3 actions are assigned to this message, which are respectively the coordinates X and Y, and the size of the dot, S. A size equal to 0 means that the dot is not tracked.

## Advanced control

Artists that create installations may want to automate the user-interface of OSCulator using OSC messages and control various functions like querying about the number of Wiimotes connected, and get their current battery charge.

OSCulator responds to a short set of OSC messages to control the Wiimote driver. Those messages must be sent to the OSC input port. Any reply will be sent to the Default Host (see OSC Routing).

The generic syntax is the following:

```
/osculator/<item> "<command-name>"
```

Where `item` designates a resource that we would like to control. For example `discovery`, is related to the discovery process. `command-name` is an OSC string that represents a command name associated with the item.

Command names ending with the character '?' are queries. After a query is received, OSCulator replies with a message that has the same OSC address, the same command name but without a question mark, and an answer argument depending on the query.

This is used as follows:

Message	Command	Description
<code>/osculator/wii/&lt;wii-id&gt;</code>		Note: <wii-id> is a number between 1 and 8, or the asterisk character '*' in which case the command matches all Wiimotes.
	<code>reset</code>	Resets the information related to the Wiimote (bluetooth address, smoothing, etc.)
	<code>battery?</code>	Queries the charge level.  The answer argument is a float number between 0.0 and 1.0 if the Wiimote is connected. If the Wiimote is not connected, the argument is the OSC value <code>Nil</code> .

Message	Command	Description
	connected?	<p>Queries the connection status.</p> <p>The answer argument is the OSC value True or False depending on the connection status.</p>

For an example of how to easily integrate this system in Max/MSP, check out the General Command Syntax file in the Wii-mote folder.



# Space Navigator

## Overview

The SpaceNavigator is a 3D mouse built by [3Dconnexion](#). It is rather inexpensive for the great sensitivity it provides. Coupled to OSCulator, it can be used as a precise surround audio mixing controller.

Six axis are available: the translations (X, Y and Z) and the rotations ( $R_x$ ,  $R_y$  and  $R_z$ ). There are also two side buttons.

No configuration is needed to use the SpaceNavigator, just plug and play.  
Several SpaceNavigators can be used at the same time.

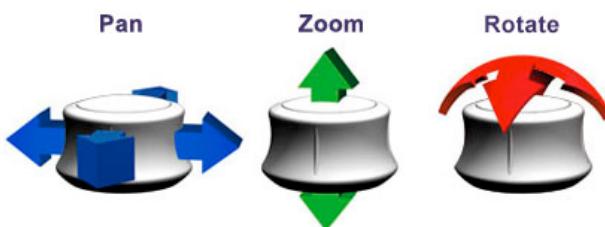


## Space Navigator messages

The messages for the SpaceNavigator are:

(with **id** being the slot under which the SpaceNavigator is connected)

- ▶ `/sp/<id>/button/1` 1 argument, left button
- ▶ `/sp/<id>/button/2` 1 argument, right button
- ▶ `/sp/<id>/rot/xyz` 3 arguments, amount of rotation on each axis
- ▶ `/sp/<id>/trans/xyz` 3 arguments, amount of translation on each axis



## Calibration

From time to time, if you notice an odd behaviour, it might be necessary to calibrate the SpaceNavigator. Open the System Preferences, and click on the 3Dconnexion Preference Panel, then click the Calibrate button and follow the instructions.



# TUIO

25



TUIO is a simple yet versatile protocol designed specifically to meet the requirements of table-top tangible user interfaces. Inspired by the idea of interconnecting various existing table interfaces such as the [reacTable](#), being developed in Barcelona and the tDesk from [Bielefeld University](#), this protocol defines common properties of controller objects on the table surface as well as of finger and hand gestures performed by the user.

Currently this protocol has been implemented within a fiducial marker-based computer vision engine developed for the reacTable project. The TUIO protocol has been implemented using [OpenSound Control](#) and is therefore usable on any platform supporting this protocol.<sup>25</sup>

While the reacTable doesn't use OSCulator, it is possible to use the video pattern recognition software [reacTIVision](#) with OSCulator very easily.

## TUIO enabled software

Here is a non exhaustive list of software using the TUIO protocol:

- ▶ [OSCemote](#) is a multi-touch controller for the iPhone developed by Joshua Minor ;
- ▶ [MSARemote](#), another multi-touch controller for the iPhone developed by Mehmet Akten ;
- ▶ [reacTIVision](#), the reacTable computer vision framework with a complete TUIO implementation ;
- ▶ [Touchlib](#) is a library for creating multi-touch interaction surfaces ;
- ▶ [TuioKinect](#), tracks simple hand gestures using the Kinect controller ;
- ▶ [tuipad](#), an open source TUIO tracker for iOS devices such as the iPad, iPhone and iPod touch.

TUIO is a very versatile protocol. Being flexible and rich in its expression, it is also a bit complicated to handle. OSCulator includes an interpreter that eases the use of TUIO.

<sup>25</sup> picture of the reacTable courtesy of [Xavier Sibecas](#)

<sup>26</sup> from <http://tuio.org>

# TUIO messages

The TUIO protocol keeps tracks of Cursors and Objects. Cursor are used to track the position and speed of a pointer, a finger for example ; Objects, are used to track the position, orientation and speed.



## Example: reacTIVision Setup

There is not much to do than launch the reacTIVision program. This application send OSC message to port 3333 on the local host by default, this means that you will need to change the OSC Input Port in OSCulator to 3333.

## Messages definition

As soon as OSCulator receives TUIO messages, it will display them in the main window<sup>27</sup>.

Those messages always come in pairs:

- the message carrying position, orientation and speed information, for example /tuio/2D/obj/1 ;
- and the message carrying activity information, for example /tuio/2D/obj/1/activity. This message tells whether or not the object is visible. When visible, a float value of 1.0 is sent, else the value is 0.0.

The position message has the following format:

/tuio/{2D,3D}/{cur|obj}/<**id**>

and the activity message:

/tuio/{2D,3D}/{cur|obj}/<**id**>/activity

## Arguments

The activity message has only one argument, the activity, being 0 (not visible or not active) or 1 (visible or active). The position message has the following arguments:

2D cursors	2D objects
/tuio/2D/cur/< <b>id</b> >	/tuio/2D/obj/< <b>id</b> >
x x position	x x position
y y position	y y position
vx speed of x axis	rz orientation (angle of rotation around z)
vy speed of y axis	vx speed of x axis
a motion acceleration	vy speed of y axis
	vrz speed of rotation
	a motion acceleration
	arz rotation acceleration



<sup>27</sup> remember, the messages are interpreted, not displayed in the raw TUIO format

# Wacom Tablet



The Wacom tablet is a device primarily used for graphic design. It simulates a virtual sheet of paper on which the user can draw with specialized tools.

OSCulator can use several tablets each having several tools. Most attributes are recognised, like pressure, pen tilt, buttons, Express Keys, Touch Strips, etc. Intuos 4, Intuos 3, Bamboo and Graphire models are known to work.

## Quick Start

- ▶ Download and install the latest driver from [Wacom's website](#).
- ▶ Press the Caps-Lock ⇧ key (messages will appear in the main window, and the mouse pointer will be locked).
- ▶ Configure the messages with events.

You can choose the way the mouse is locked by changing the Preferences (see Wacom Preferences).

## Wacom tablet messages

When the tablet is activated, the messages displayed in the main window all bear in common the /wacom prefix, followed by the tablet number. For example: /wacom/1/... (where ... is the rest of the message name).

After the tablet number, an identifier is used for the tool type:

- ▶ pen for the Pen tool thin tip side
- ▶ eraser for the Pen tool eraser side
- ▶ puck for the mouse pointer
- ▶ strip for Touch Strips
- ▶ key for Express Keys

The Pen and the mouse being movable tools, a proximity message is sent when they leave or enter the proximity range. Such message is named for example: /wacom/1/pen/0/proximity. Such message could be used to trigger a note when the tool is near the tablet, and stopping the note when it goes away.

Tools that have buttons send additional messages. For example /wacom/1/pen/0/button/1 refers to the first button of the pen 0 on tablet 1.

## Goodies

The Sample Library folder contains presets for controlling [Plogue's Bidule](#) or [Max/MSP](#) from the Wacom Tablet.



# Appendices

## Event input range

Most events accept values between 0.0 and 1.0, but there are a few exceptions. The following table lists the expected input range for each event type. For most events, the input range can be customised by adjusting the Input Min and Input Max parameters in the [Scalings page](#).

Event Type	Input Range	
	Float	Integer
MIDI CC	[0.0 1.0]	[0 127]
MIDI CC Toggle	[0.0 1.0]	[0 1]
MIDI Prg	[0.0 1.0]	[0 1]
MIDI Prg (Change by Value)	[0.0 1.0]	[0 127]
MIDI Note	[0.0 1.0]	[0 1]
MIDI Note (Channel Velocity)	[0.0 1.0]	[0 127]
MIDI Note (Channel Pressure)	[0.0 1.0]	[0 127]
MIDI Note (Pitch Bend)	[0.0 1.0]	[0 16383]
MIDI Note w/ Params	[0.0 1.0]	[0 1]
Kyma CC	[0.0 1.0]	[0 1]
Kyma CC Toggle	[0.0 1.0]	[0 1]
Kyma Prg	[0.0 1.0]	[0 1]
Kyma Note	[0.0 1.0]	[0 1]
Kyma Note w/ Params	[0.0 1.0]	[0 1]
Note Params (Pitch, Velocity, Aftertouch)	[0.0 1.0]	[0 127]
Note Params (Voice)	integer  The voice index can be any integer. Floating numbers are transparently coerced to integers.	[0 n-1]
OSC Routing	any	any
Keycode	[0.0 1.0]	[0 1]
Key Combo	[0.0 1.0]	[0 1]
Mouse (Buttons, Scroll Wheel)	[0.0 1.0]	[0 1]

[x y] is used to represent a range – 1.0 denotes a floating number – 128 an integer – any, any kind of OSC value (number, string, etc.) – integer, any integer like -15, 0, 42, etc. – positive integer, an integer equal to or higher than 0.

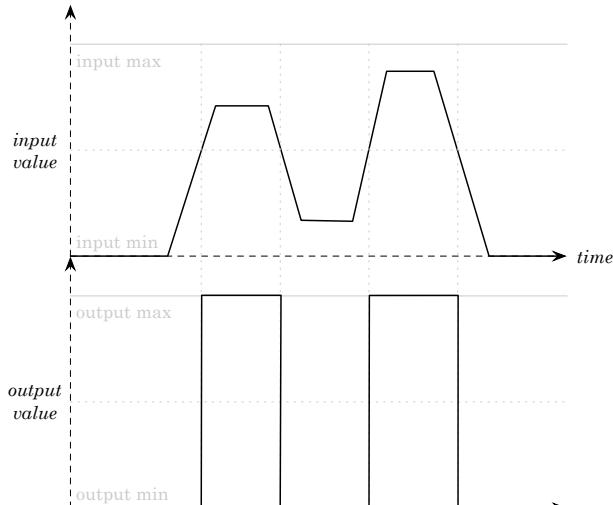
Event Type	Input Range	
	Float	Integer
Mouse Relative Move	[0.0 1.0]	n/a
Mouse Absolute Move	[0.0 1.0]	[0 n] Where n is the width or height of the screen in pixels. For multiple screens, all widths or heights are summed.
HID (Axes, Buttons, Analog)	[0.0 1.0]	[0 1]
AppleScript	[0.0 1.0]	[0 1]
Console Log	any	any
Enable	[0.0 1.0]	[0 1]
Enable Toggle	[0.0 1.0]	[0 1]
Enable Combine	[0.0 1.0]	[0 1]
Presets (Next, Previous)	[0.0 1.0]	[0 1]
Presets (Change by Index)	positive integer <small>0 refers to the first preset and n-1 the last preset.</small>	[0 n-1]
Preset (Switch to...)	[0.0 1.0]	[0 1]
Change Channel	[0.0 1.0]	[0 15]
Variable	any	any
Wiimote Enable	[0.0 1.0]	[0 1]
Wiimote Vibrate	[0.0 1.0]	[0 1]
Wiimote LEDs (Numeric)	[0.0 16.0]	[0 16]
Wiimote LEDs (Level)	[0.0 1.0]	[0 1]
Wiimote Motion Plus Reset	[0.0 1.0]	[0 1]

# Event triggering

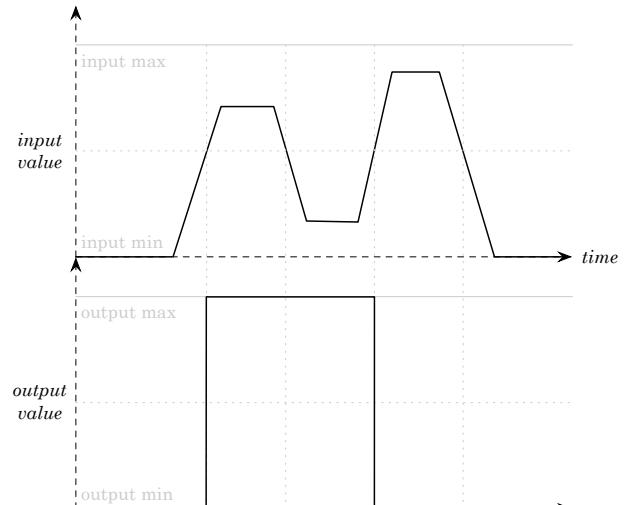
Some Events process their input to automatically accommodate any numerical value. For example, the MIDI Program Change event only sends the Program Change message when the value changes from a value higher than 0.5, to a value lower than 0.5.

The triggering point is defined as the middle value between Input Min and Input Max (found in the [Scalings page](#)).

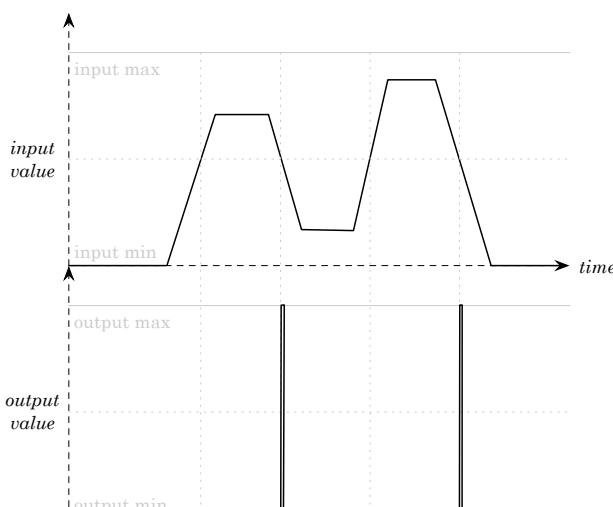
## Trigger types



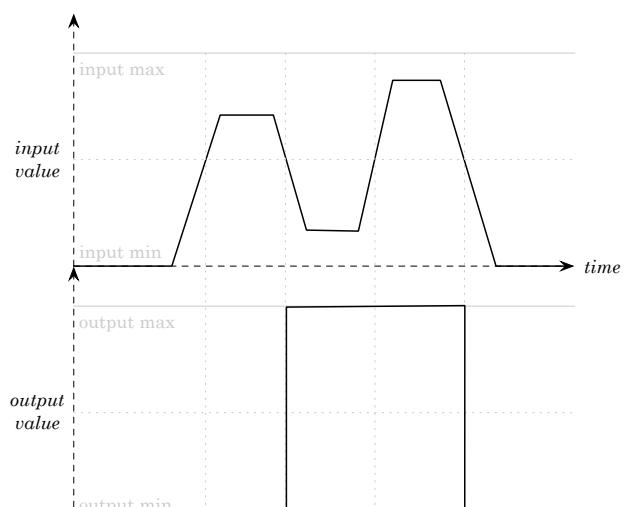
Standard Trigger



Toggle (up)



Trigger (down only)



Toggle (down)

Note: Trigger (down only) is used on events that happen at once, like a MIDI Program Change or a Preset Change. Their triggering is best when the value decreases. Practically, when the signal comes from a touch control surface application like TouchOSC, the trigger is activated when the finger releases a push button.

In this table, the terms Input Min, Input Max, Output Min and Output Max refer to parameters of the [Scalings page](#).

Event Type	Trigger Type	Notes
MIDI CC	No Trigger	A MIDI Control Change is sent every time numerical input is received, even if two consecutive values are identical
MIDI CC Toggle	Toggle	A MIDI Control Change message is sent when the input crosses upwards the middle value between Input Min and Input Max. The output value alternates between Output Min or Output Max.
Kyma CC Toggle	Toggle	A Kyma Control Change message is sent when the input crosses upwards the middle value between Input Min and Input Max. The output value alternates between Output Min or Output Max.
MIDI Prg	Trigger (down only)	A MIDI Program Change message is sent when the input crosses downwards the middle value between Input Min and Input Max.
Kyma Prg	Trigger (down only)	A Kyma Program Change message is sent when the input crosses downwards the middle value between Input Min and Input Max.
MIDI Note	Standard Trigger	A MIDI Note message is sent when the input crosses the middle value between Input Min and Input Max. A Note On is sent when the value goes up, and a Note Off is sent when the value goes down. Output Min and Output Max parameters are ignored.
Kyma Note	Standard Trigger	A Kyma Note message is sent when the input crosses the middle value between Input Min and Input Max. A Note On is sent when the value goes up, and a Note Off is sent when the value goes down. Output Min and Output Max parameters are ignored.
MIDI Note w/ Params	Standard Trigger	A MIDI Note message is sent when the input crosses the middle value between Input Min and Input Max. A Note On is sent when the value goes up, and a Note Off is sent when the value goes down. Output Min and Output Max parameters are ignored.
Kyma Note w/ Params	Standard Trigger	A Kyma Note message is sent when the input crosses the middle value between Input Min and Input Max. A Note On is sent when the value goes up, and a Note Off is sent when the value goes down. Output Min and Output Max parameters are ignored.
Keycode	Standard Trigger	The key is pressed when the input crosses upwards the middle value between Input Min and Input Max. The key is released when the value goes downwards. Output Min and Output Max parameters are ignored.
Key Combo	Standard Trigger	The key combo is pressed when the input crosses upwards the middle value between Input Min and Input Max. The key combo is released when the value goes downwards. Output Min and Output Max parameters are ignored.

In this table, the terms Input Min, Input Max, Output Min and Output Max refer to parameters of the [Scalings page](#).

<b>Event Type</b>	<b>Trigger Type</b>	<b>Notes</b>
Mouse (buttons)	Standard Trigger	The mouse button is pressed when the input crosses upwards the middle value between Input Min and Input Max. The mouse button is released when the value goes downwards.  Output Min and Output Max parameters are ignored.
HID (buttons)	Standard Trigger	The HID button is pressed when the input crosses upwards the middle value between Input Min and Input Max. The HID button is released when the value goes downwards.  Output Min and Output Max parameters are ignored.
AppleScript	Trigger (down only)	The AppleScript is executed when the input crosses downwards the middle value between Input Min and Input Max.  Output Min and Output Max parameters are ignored.
Enable	Standard Trigger	The target message is enabled when the input is higher than the middle value between Input Min and Input Max.  Output Min and Output Max parameters are ignored.
Enable Toggle	Toggle	The target message is alternatively enabled or disabled when the input crosses upwards the middle value between Input Min and Input Max.  Output Min and Output Max parameters are ignored.
Enable Combine	Standard Trigger	The target message is enabled when the input is higher than the middle value between Input Min and Input Max.  Output Min and Output Max parameters are ignored.
Preset Change	Trigger (down only)	The preset is activated when the input crosses downwards the middle value between Input Min and Input Max.  Output Min and Output Max parameters are ignored.
Wiimote Enable	Standard Trigger	The Wiimote is enabled when the input is higher than the middle value between Input Min and Input Max.  Output Min and Output Max parameters are ignored.
Wiimote Vibrate	Standard Trigger	The Wiimote starts vibrating when the input is higher than the middle value between Input Min and Input Max.  Output Min and Output Max parameters are ignored.
Wiimote Motion Plus Reset	Trigger (up only)	The Wiimote MotionPlus is reset when the input crosses upwards the middle between Input Min and Input Max.  Output Min and Output Max parameters are ignored.

# Legal & Copy Notices

Written by Camille Troillard.

For any inquiry or suggestion, please contact [camille@osculator.net](mailto:camille@osculator.net).

Content of this manual is copyright **Wildora** © 2010, 2011, 2012.

OSCulator and the OSCulator logo are trademarks of Wildora S.à.R.L, registered in France and E.U., U.S. registration pending. Mac, the Mac logo and the Bonjour logo are trademarks of Apple Computer, Inc., registered in the U.S. and other countries. All other trademarks are the property of their respective owners.

