

Lemur



V2.0  
USER GUIDE

<b>Chapter 1 - Introduction.....</b>	<b>6</b>
1.1 Welcome .....	6
1.2 What's new in V2.0 ? .....	8
<b>Chapter 2 - Concepts.....</b>	<b>10</b>
2.1 The JazzEditor .....	10
2.2 Projects.....	10
2.3 Interfaces .....	10
2.4 Modules.....	11
2.5 Objects.....	12
2.6 Variables.....	13
2.7 Scripts .....	13
2.8 OSC .....	13
2.9 MIDI.....	13
2.10 Keyboard and Mouse.....	14
2.11 JazzDaemon.....	14
2.12 Targets.....	14
<b>Chapter 3 - Hardware &amp; Setup.....</b>	<b>15</b>
3.1 Anatomy .....	15
3.2 Network configuration .....	16
3.3 Direct connection in DHCP mode.....	17
3.4 Right Touch.....	18
3.5 Recommendations.....	18
<b>Chapter 4 - Software Installation .....</b>	<b>19</b>
4.1 Mac OSX Install .....	19
4.2 The JazzDaemon on MacOSX.....	20
4.3 Windows Install .....	22
4.4 The JazzDaemon on Windows.....	24
<b>Chapter 5 - Introducing the JazzEditor .....</b>	<b>26</b>
5.1 Overview.....	26
5.2 Header .....	27
5.3 Lemur Panel.....	28
5.4 Creation Panel.....	30
5.5 Project Panel.....	32
5.6 Script Panel .....	35
5.7 Objects Panel .....	38
5.8 Mapping Panel.....	40
5.9 Settings Menu.....	42
<b>Chapter 6 - First Steps .....</b>	<b>45</b>
6.1 Connecting your Lemur .....	45
6.2 Creating an Interface.....	46

6.3	Creating Objects .....	47
6.4	Saving your Project .....	48
6.5	The Lemur's memory for Projects.....	49
6.6	Changing Object Appearance .....	50
6.7	Groups .....	51
6.8	Configuring Object's Behaviors.....	54
6.9	Using Containers.....	58
6.10	Import and Export of Modules .....	64
<b>Chapter 7 - Mapping .....</b>		<b>65</b>
7.1	Setting up MIDI messages.....	65
7.2	Simple MIDI mapping examples .....	68
7.3	OSC .....	72
7.4	Keyboard and Mouse .....	74
7.5	Trigger Modes.....	76
<b>Chapter 8 - Targets Setup .....</b>		<b>77</b>
8.1	JazzDaemon Targets .....	77
8.2	JazzDaemon Targets Setup Example .....	79
8.3	OSC Targets .....	86
<b>Chapter 9 - Going further with the JazzEditor .....</b>		<b>88</b>
9.1	Control your Objects with your Objects.....	88
9.2	Making your own Variables.....	90
9.3	Using Vector Variables.....	93
9.4	Using Custom MIDI Messages.....	96
9.5	Bidirectional Control.....	98
9.6	Defining and Using Functions .....	99
<b>Chapter 10 - Introducing Multi-line Scripts .....</b>		<b>101</b>
10.1	Creating a Script .....	101
10.2	Execution.....	102
10.3	Variables .....	103
10.4	Attributes .....	104
10.5	Built-in Functions and Operators .....	106
10.6	Examples .....	106
<b>Chapter 11 - Advanced Scripting.....</b>		<b>112</b>
11.1	Conditional statements .....	112
11.2	Loops .....	115
11.3	Return .....	118
<b>Chapter 12 - Object Reference .....</b>		<b>120</b>
12.1	Breakpoint .....	120
12.2	Container.....	123
12.3	Custom Button.....	124
12.4	Fader .....	126

12.5	Gesture.....	129
12.6	Knob.....	131
12.7	Leds .....	134
12.8	LemurMenu.....	136
12.9	Monitor .....	138
12.10	MultiBall .....	140
12.11	MultiSlider .....	144
12.12	Pads .....	146
12.13	Range .....	149
12.14	RingArea.....	151
12.15	SignalScope .....	153
12.16	SurfaceLCD.....	155
12.17	Switches .....	156
12.18	Text .....	158
<b>Chapter 13 - Parser Reference</b>	.....	<b>159</b>
13.1	Vectors and Singletons .....	159
13.2	Lemur Internal clock .....	160
13.3	Built-in Time Variable.....	160
13.4	Built-in current_interface Variable.....	161
13.5	Built-in midi_clocks Variable .....	161
13.6	Arithmetic Functions .....	162
13.7	Object-related Functions .....	164
13.8	Scripting Output Functions.....	167
13.9	Vectorial Functions.....	168
13.10	Trigonometric functions .....	171
13.11	Operators.....	171
<b>Chapter 14 - MIDI Message Reference</b>	.....	<b>175</b>
14.1	Note Off .....	175
14.2	Note On .....	175
14.3	Key Pressure (Polyphonic Aftertouch).....	176
14.4	Control Change.....	177
14.5	Program Change .....	177
14.6	Channel Pressure.....	178
14.7	Pitch Bend .....	178
14.8	System Exclusive .....	179
14.9	Song Position.....	179
14.10	Song Select .....	180
14.11	Bus Select.....	180
14.12	Tune Request .....	180
14.13	Timing Tick.....	181
14.14	Start Song .....	181

14.15 Continue Song .....	181
14.16 Stop Song .....	181
14.17 Active Sensing .....	182
14.18 System Reset .....	182
<b>Appendix I - Network Configuration Guide.....</b>	<b>183</b>
AI.1 Glossary .....	183
AI.2 Direct Connection .....	183
AI.3 Computers and Lemur connected to a Network .....	184
AI.4 Lemur Network Setup .....	184
AI.5 MacOS Network Setup.....	186
AI.6 Windows XP Network Setup .....	190
<b>Appendix II - Hardware General Settings.....</b>	<b>194</b>
<b>Appendix III - FTP Server .....</b>	<b>195</b>
<b>Appendix IV - Shortcuts.....</b>	<b>196</b>
<b>Appendix V - Warranty .....</b>	<b>197</b>
Warranty and Liability.....	197
Your Rights.....	197
Applicable Laws and Court of Jurisdiction.....	197

# Chapter 1 - Introduction

## 1.1 Welcome

Congratulations for choosing JazzMutant's Lemur! The Lemur is a groundbreaking input device for controlling music and media applications. Its unprecedented modularity combined with its unique Multitouch sensing technology will forever change the way you compose, perform, produce and, in more general terms, interact with your work environment.

### What the Lemur is

The Lemur is a computer peripheral. In other words, it is a device that you connect to a computer.

The Lemur is a controller: its only purpose in life is to provide hands-on control over software running on a computer, or other devices connected to it - such as samplers, synthesizers etc. To interact with them, the Lemur uses two communication protocols: MIDI and Open Sound Control. Software must comply with at least one of those protocols to be controlled by the Lemur.

### ... and what it is not

The Lemur is neither a tablet PC nor a display monitor. That means you cannot write emails or watch a movie with it.

The Lemur does not produce any sound by itself. There is plenty of wonderful software and hardware out there perfectly doing that job. The Lemur is there to make the use of that software and hardware more efficient and hands-on.

The Lemur does not feature a physical Midi Interface; you cannot directly connect it to a master keyboard or any other MIDI device.

In short, the Lemur requires that you own at least one computer running at least one compatible application in order to work. In case you don't, the Lemur is ultimately worthless.

## About this Manual

Considering the great variety of applications that your Lemur can control, it would be nearly impossible to exhaustively detail here all the interfacing possibilities. Therefore, instead of providing step-by-step tutorials for every single piece of software you might use, this manual focuses on the general features and concepts one needs to master in order to work with the Lemur. Besides this manual, there exists a host of external material, including additional documentation, templates and didactical workshops meant to make your introduction to Lemur work as smooth as possible. Some of it is available on the installation CD and a lot more can be downloaded from <http://www.jazzmutant.com>.

Last, another great characteristic of the Lemur is the constant evolution of its software, always listening to the Lemur community's needs. For that purpose, JazzMutant regularly releases new updates to both software and documentation. Consequently, the manual you are reading might already be out-of-date. Before getting deeper into the documentation, please make sure that the present documentation matches the software version installed on your Lemur. In general, we suggest keeping both your Lemur and the above documentation up-to-date by regularly checking the product section on JazzMutant's website.

## 1.2 What's new in V2.0 ?

### 1.2.1 The JazzEditor's Palette

This panel allows you to drag objects from the icons list and drop them directly onto the work area. Similarly, you may also drag modules from the new Library panel provided you've specified a location for your modules library in the JazzEditor's settings. Inversely you may also alt-click on a selection and drag it to the library to save it as a module, or drag it anywhere on the canvas to duplicate it.

### 1.2.2 The concept of Parent

When affecting MIDI or OSC messages to variables, it's no longer necessary to manually enter the Target for each variable individually. By default, a variable's Target is set to that of the object to which it belongs, its Parent. A Target can also be set to an entire project, and the targets of the objects it contains automatically set to the project's target.

### 1.2.3 Extended Scripting Abilities

The Script panel's purpose is now two-fold: it allows for the single-line definition of User-defined Variables, or the multi-line coding of a Script created with the Create Script icon. Multi-line scripts can be used, in conjunction with a set of powerful new functions, to instruct your Lemur to perform specific tasks at specific times. A typical application of this new feature will be the real-time manipulation of an object's attributes, such as its dimensions, colour or physics. A basic example would be to instruct a Switch to activate a Multiball's grid, or a Text object to change its content upon reception of an OSC or MIDI message.

### 1.2.4 Mouse and Keyboard Control

The JazzEditor's mapping panel now features a Keyboard and mouse tab, allowing you to assign any key commands, mouse movements and mouse clicks to your Objects. For that purpose, you'll also notice a new dedicated *KbMouse* destination amongst your Targets. This is also where scripts containing the new functions such as *keystate*, *mousemove* and *mouselbutton* will output their operations.

### 1.2.5 Tabbed Container Objects

A tabbed Container can be created simply by ctrl-clicking/ right-clicking onto a standard Container object and choosing the option *Make tabbed*. Additional tabs can then be inserted with the option *Add tab*. The tabs' order can be modified by clicking within a Container and choosing the *Bring up* or *Bring down* options.

## 1.2.6 Aliases

An alias is a perfect replica of an object, behaving exactly as the original object and in turn making the original object and other aliases behaving as itself. Ctrl-click/ right-click on an object to create an alias of this object.

## 1.2.7 The Breakpoint Object

The Breakpoint is a multi-segment envelope editor. It can display up to 64 points, which you may either define in the object's properties, or, if the *Live Edition* option is activated, create in real-time by double-touching the object's surface. By default the object constraints a point's horizontal displacement so that it does not travel further than its left neighbour ( $x[2] \geq x[1] \geq x[0]$ ), but that can be deactivated by ticking the *Free Form* option. It also features a *Super-Spring* Physics mode, which makes segments springy. In this mode, the distance between each point can be set to a fixed value with the *Rest* parameter.

## 1.2.8 The Gesture Object

A trackpad control with advanced gesture recognition such as pinch, rotation and traces. Its x and y variables react like those of a traditional trackpad, taking acceleration into account. Overall, the purpose of this object is to provide alternative and hopefully intuitive ways of controlling distinct elements from a single object.

## 1.2.9 The LemurMenu Object

A new “pop-up” menu selection tool. It can hide and display a list of up to 32 individual items. By default the object outputs the index of the selection, but this can be MIDI scaled with the *Scale Output* option.

## 1.2.10 Multi-mode Colour Picker

And last but not least, a new, sexier looking JazzEditor with a 2D colour picker. You can select different colour pickers by ctrl-clicking/ right-clicking and choosing between the RGB, HSV, Hue or Grayscale colour pickers. The small squared color sample you're left with when you've picked a colour can be dragged to other objects on the workspace for quick matching.

# Chapter 2 - Concepts

The Lemur introduces a new paradigm in the realm of media control, and along with this brand new approach, its usage might slightly differ from the way you are used to deal with conventional controllers. Therefore, before you start working with your new controller, we felt it was worth making you familiar with the core concepts of the system.

## 2.1 The JazzEditor

As you might have noticed, the main characteristic of the Lemur is that it doesn't feature any physical control object, such as real pads, keyboard, bulky faders or joysticks. Instead, there is a blank multitouch panel that you can populate with as many different virtual objects as you need. This way, you can design the interface layouts that will perfectly fit both your application and your hands. For that purpose, the Lemur is provided with a dual-platform (MacOSX and Windows) interface design utility, the mighty JazzEditor.

## 2.2 Projects

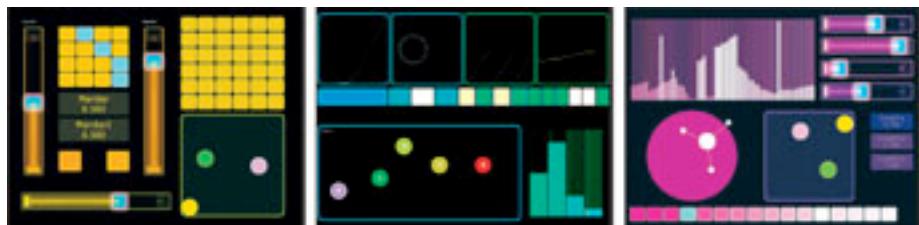
The JazzEditor lets you import or create complete control templates for the Lemur, which we call Projects. All Project files the JazzEditor produces can be stored on your computer's hard drive as JZML files.

Projects can be stored in the Lemur's internal Flash memory too. You can access the Lemur Project browser with the second button on your Lemur and a touch on the Project Browser button.

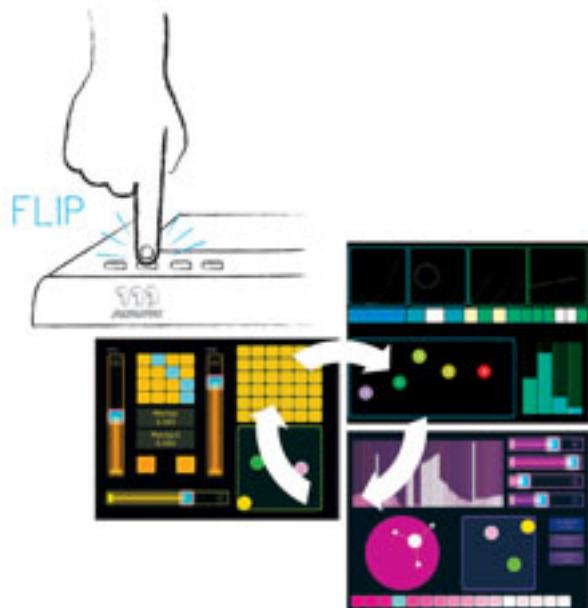
## 2.3 Interfaces

A Project can contain many Interfaces, which you can think of as the pages of your Project. Interfaces contain in turn multiple Objects, which can also be organized in Containers if desired. The number of Interfaces and Objects is limited only by the internal memory of the Lemur, which gives you plenty of room for your creations.

The figure on the next page shows a typical Lemur project, comprised of three different interfaces:



Once a project is loaded on the Lemur, the different Interfaces can be browsed through via the Interface List button or switched to with the Navigation buttons. The Interface List button presents them as an array of small icons that you can activate by touch. Use the Navigation buttons to step through the Interfaces one by one, as shown below.



## 2.4 Modules

Additionally, you can import pre-built parts of project as Modules. A Module is a set of reusable interface elements saved as JZLIB files. A module can feature objects, containers and even interfaces. A library of modules is provided on the Installation CD, and more modules are available from JazzMutant's website. You can also create your own user library of reusable items. Indeed, any part of a Lemur Interface can be exported as a Module for future use in other Projects. Thanks to this feature you never have to develop the same thing twice. Just save your building blocks as Modules to produce a growing library of re-useable, object-oriented Interface parts.

The User Area on JazzMutant 's website is also a great resource for stacking up on handy Interface elements.

## 2.5 Objects

Objects are the main message generators of the Lemur: they provide the values you control with your fingers. There is a multitude of different Objects available that all have their special capabilities:



**Breakpoint**



**LemurMenu**



**Container**



**Monitor**



**CustomButton**



**MultiSlider**



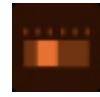
**Fader**



**Pads**



**Gesture**



**Range**



**Knob**



**RingArea**



**SurfaceLCD**



**SignalScope**



**Leds**



**Switches**



**MultiBall**



**Text**

## 2.6 Variables

Most Objects have their own set of Variables that reflect the Objects' states, and change when you touch them with your fingers. A Fader's X Variable reflects the position of its cap for example, or that of a Switch its on/off state. All these Variables can also be modified by external software, by Scripts or by other Objects dwelling in the Project.

In addition to these Built-in variables, the JazzEditor lets you create your own User-defined Variables, which you can equate to your own mathematical expressions.

A Variable can be local, living in a specific Object and accessed from the outside through its address. A Variable can also be defined globally for a Project, enabling multiple Objects to use its values directly.

Variables can be multi-dimensional too. A Knob, for instance, has only one Variable defining its output with a single number, while A MultiBall Object has three Variables: the X and Y positions of the balls, and a third Z Variable representing their Brightness. The MultiBall's Variables are Vectors with as many components as there are balls defined for the Object.

The built-in Time Variable deserves a special mention. It presents the time in milliseconds since the Lemur was switched on. Combined with the Lemur's functions and mathematical operators, it can produce a great variety of time-varying number sequences.

## 2.7 Scripts

Scripts can be used to instruct your Lemur to perform specific tasks at specific times. From simple actions such as changing a MultiSlider colour when a Switch is pressed, to complex actions such as commanding a Multiball's balls to follow set trajectories. The possibilities are endless.

## 2.8 OSC

The Lemur is one of the first controllers to comply with the Open Sound Control protocol, an emerging network-based standard with significant advantages over MIDI: low latency, higher data capacity, 32-bit numerical precision, flexibility and easy set-up. Supported by a growing number of high profile applications (Max/MSP, Reaktor, Circle, Modul8, Resolume Avenue...), OSC opens a new era in the field of real-time control and human-machine interfaces.

## 2.9 MIDI

Naturally, your Lemur also complies with the MIDI protocol and can transmit and receive all the general MIDI messages. Any of these messages can directly be assigned to any Built-in or User-defined Variables, sent through scripts, or activate a script execution upon reception by your Lemur.

## 2.10 Keyboard and Mouse

In addition to OSC and MIDI, Objects can also transmit keystrokes, mouse clicks or mouse movements to your computer via the JazzDaemon's KbMouse port. These can also be directly assigned to any Built-in or User-defined Variables or sent through scripts.

## 2.11 JazzDaemon

The JazzDaemon is a little helper application running in the background on your computer. It handles the MIDI data flow between the Lemur and the MIDI ports installed on the computer, as well as any mouse movements or ASCII commands through the KbMouse port.

The JazzDaemon automatically scans all available MIDI ports on the computer, be they physical or virtual, and make them available to the Lemur(s) connected to the network. On Mac, the JazzDaemon also creates its own set of virtual ports, the Daemon inputs and outputs, which are accessible in your applications' MIDI settings. On PC, third-party virtual ports such as those offered by MIDI Yoke or Maple are needed in order to link to the JazzDaemon and use MIDI with your Lemur.

## 2.12 Targets

The networkability of the Lemur endows it with the unique capacity to simultaneously control several applications running on the same computer, or even to control several pieces of software launched on different computers. Each application or device controlled by the Lemur being represented by a "Target". The Lemur features eight JazzDaemon Targets and eight OSC Targets.

A JazzDaemon Target consists of a pair of MIDI In- and Outputs together with a KbMouse Output for keyboard and mouse commands. An OSC Target is characterised by its IP address and its port number. No matter how complex your setup may be, you can control everything from a single Interface on your Lemur by individually assigning the different Variables to different Targets, or by using Scripts containing Output functions.

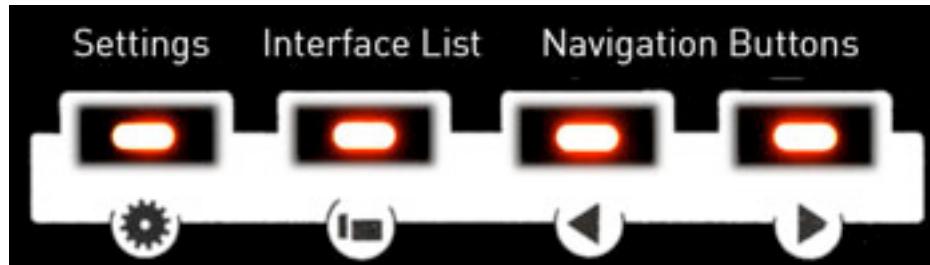
You can connect to the MIDI ports of any computer on the network provided they have a JazzDaemon running. Use the JazzDaemon Targets settings page on your Lemur to get a list of running JazzDaemon's on the network and to connect to specific MIDI ports on the respective machines.

# Chapter 3 - Hardware & Setup

## 3.1 Anatomy



The top of the Lemur obviously features the Multitouch screen, consisting of the touch-sensitive surface and the ultra-bright TFT Display. The first allows you to manipulate colourful user interface objects displayed on the latter with all your ten fingers independently. Above the display, there are four buttons that have different utility functions:



- The **Settings** button gives you access to your Lemur network settings, JazzDaemon and OSC Targets, as well as screen and touch panel configuration. It also displays the firmware version installed on your machine.
- The **Interface List** button displays a gallery of thumbnails depicting each interface currently stored on the unit. Touch the desired interface to display it full screen. The Interface List page also contains a link to the Lemur **Project browser**.
- The two **Navigation** buttons allow you to move forwards and backwards through the Interfaces currently stored on the unit.

On the rear panel of the Lemur, you find the power switch and the power input jack.

Only connect the original power supply that came with your unit, as it is perfectly adapted to the Lemur, or a power supply that exactly matches the characteristics of the original power supply.

You'll find the Ethernet port with its two status lights next to the adapter input: this is the Lemur's door to communication with the outside world.

## 3.2 Network configuration

The Lemur uses standard Ethernet to communicate with your computer or with other devices on the network. You can connect it directly to your computer with a single crossover Ethernet cable, or place it on a local area network via a hub or switch and a non-crossover Ethernet cable.

You may want to use a local area network if any of the following are true:

- You are using your computer's Ethernet port to connect to the internet, and you want to keep the connection while working with your Lemur.
- You use a WLAN connection to the internet via a wireless router, and you would like to access your Lemur wirelessly by connecting it to your router.
- You have several Lemurs you would like to use in parallel (Lemurs are very social creatures – one is never enough).

You may want to connect your Lemur directly to your computer with a single cable if any of the following are true:

- Your Lemur is the only Ethernet device you have (either because you don't own any others or you connect wirelessly to other Ethernet network devices).
- You are looking for the simplest possible configuration for live performance.

To get in business rapidly, we'll only cover the latter configuration in this chapter. It should suffice on most systems. Should it not work out of the box, or if you seek more details on OS tinkering and connection to hubs and switches, please have a look at the Appendix on Network Configuration.

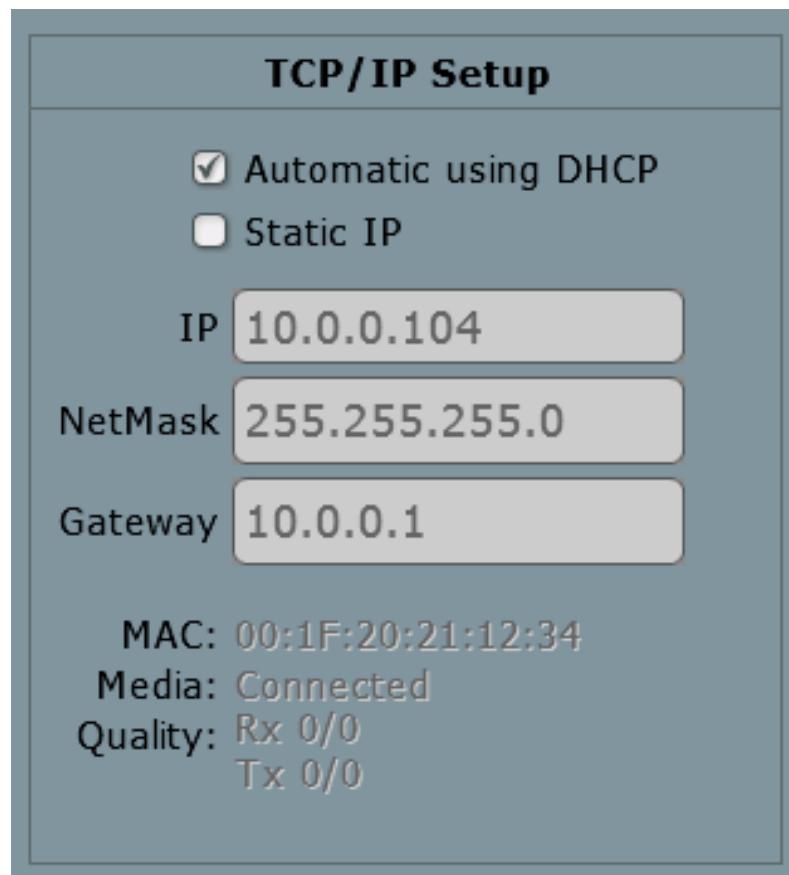
### 3.3 Direct connection in DHCP mode

The direct connection procedure basically boils down to three steps.

- Connect the Lemur's Ethernet port to your computer's one, using the provided crossover cable.



- Make sure any firewall software is deactivated on your computer.
- Turn your Lemur on and press the settings button, and select the **Automatic Using DHCP** mode on the TCP/IP Setup panel.



## 3.4 Right Touch

The Multitouch technology used in the Lemur combines high resolution with high sensitivity. You don't have to use excessive force to control the objects on your interfaces. Extremely light touch, on the other hand, won't work either. Practice a little with your Lemur and you will quickly find the right balance combining comfort and precision.

A good technique is to use the fingertips or even the nails rather than the flat underside of the fingers. This leads to a maximum of precision and ensures good agility for the various multi-finger gestures you will develop when working with the Lemur.

## 3.5 Recommendations

There are a few elementary rules to follow if one want to have a pleasant and durable life with one's Lemur:

- When transporting your Lemur, please ensure it is properly packed to protect it from shock and damage. A good idea would be to use a sturdy laptop case (15" or 17" should fit) and place an additional protective layer over the display.
- There are two types of power cables enclosed with your Lemur. One is of the European type and one works in the USA. If you take your Lemur abroad, please make sure you get the appropriate power cable.
- For everyday cleaning wipe the unit with a soft, dry cloth or one that has been slightly dampened with water. To remove stubborn dirt, use a cloth with a mild, non-abrasive detergent. Afterwards, be sure to wipe the unit thoroughly with a soft, dry cloth.
- Never use gasoline, thinners, alcohol or solvents of any kind. Chemicals may cause fading of colours and/or deformation of the surface.
- When working with the unit's buttons, display, or other controls or when using its cords and cables try to be reasonably gentle. Rough handling can lead to malfunctions.
- Never strike or apply strong pressure to the display.
- When connecting/disconnecting all cables, grasp the connector itself. Never pull on the cable. This way you will avoid causing shorts, or damaging the cable's internals.
- Please use CAT5 or better quality Ethernet network cables. Lower grades will not work properly. Depending on the situation you will either need a crossover network cable (for direct connection to the computer - that's the cable that came with your Lemur) or a non-crossover network cable (for connecting the Lemur to a hub or switch; this situation is described in the Appendix Network Configuration Guide).

# Chapter 4 - Software Installation

For the Lemur to become fully functional, you will need to run the Lemur Installer package provided on the Installation CD or downloaded from your JazzMutant's User account on our website.

The Lemur Installer contains the **JazzUpgrade** application, for updating your Lemur's firmware to the latest version, the **JazzEditor**, for creating and transferring your interfaces to your Lemur, and the **JazzDaemon**, a small application running in the background required to handle MIDI communication and transmit keyboard and mouse commands from your Lemur.

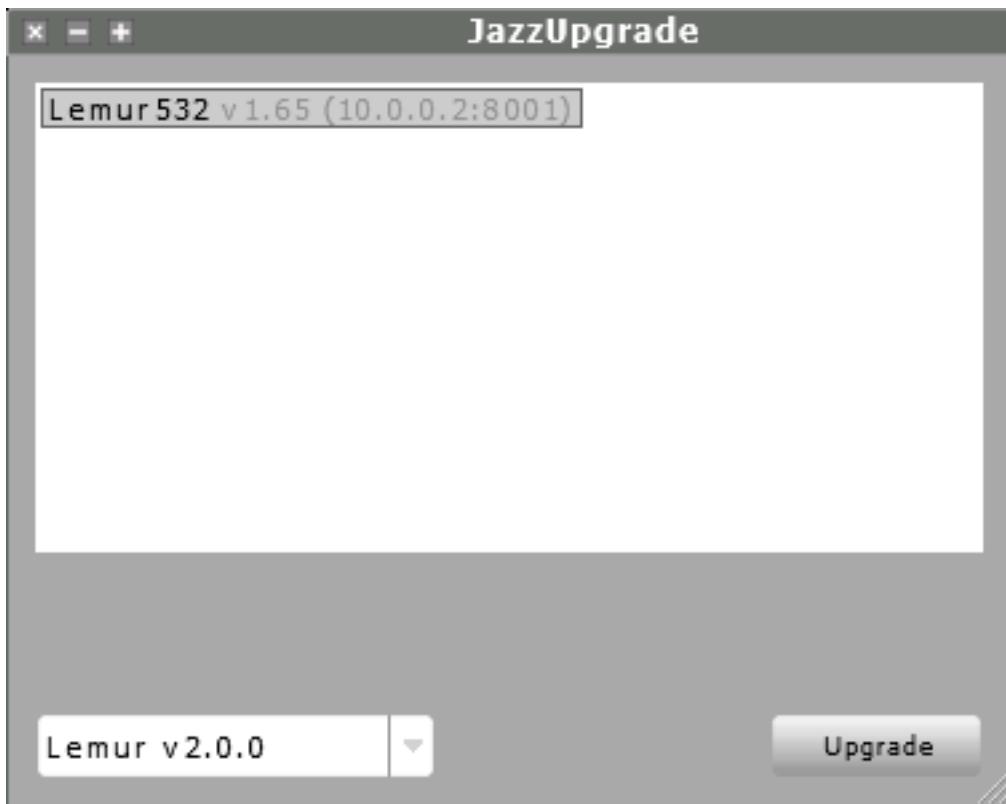
## 4.1 Mac OSX Install

Download the latest software from the web, or insert the installation CD into your Macintosh, then double-click the **Lemur v2.0.dmg** file.



Open the **Lemur v2.0 JazzUpgrade** application if your Lemur runs an older version (the version of the firmware currently installed on your Lemur is indicated on your Lemur's settings page).

The JazzUpgrade will then display a window listing the Lemur(s) connected to the network:

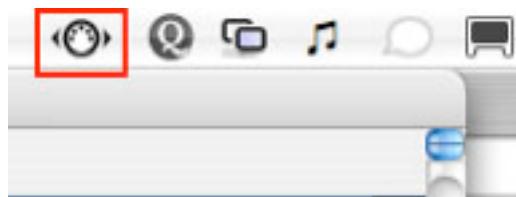


Select the Lemur you want to upgrade in the list, then click **Upgrade**. The new firmware will be uploaded to the Lemur, which will then reprogram itself and reboot with the new firmware. If the JazzUpgrade does not list your Lemur in the window, double-check your Network settings, your cable connections, deactivate any firewall you may have running and make sure any previous versions of the JazzEditor or JazzDaemon are turned off.

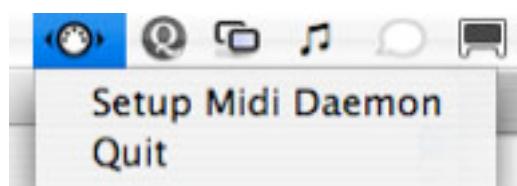
To install the **JazzEditor** and **JazzDaemon** applications, simply drop the corresponding icons in your **Applications folder**.

## 4.2 The JazzDaemon on MacOSX

The JazzDaemon doesn't appear in your Dock while it's running, but you can find it on the Menu Bar of your Mac OS X desktop.



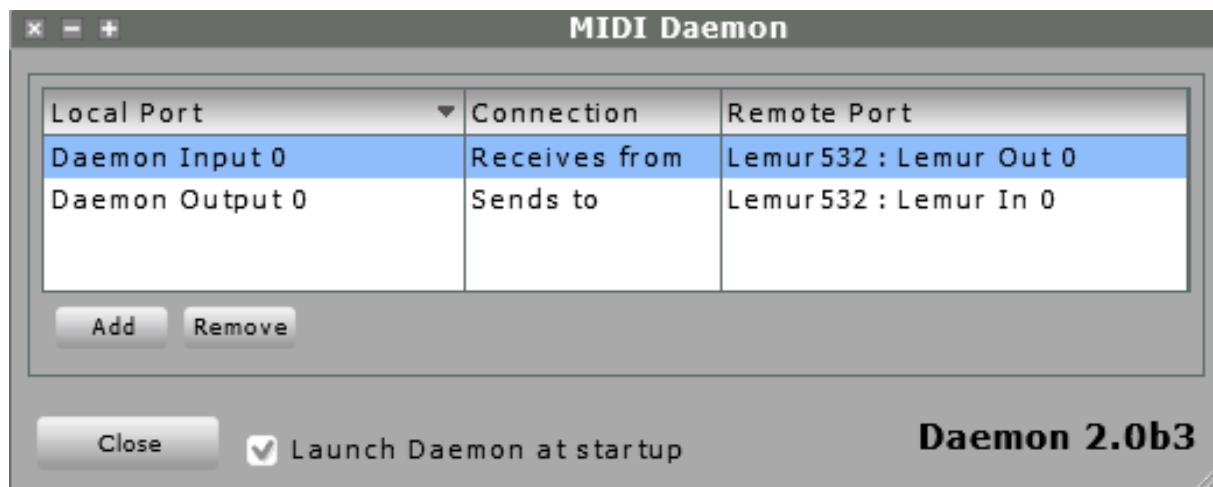
A click on the JazzDaemon opens a menu with two entries:



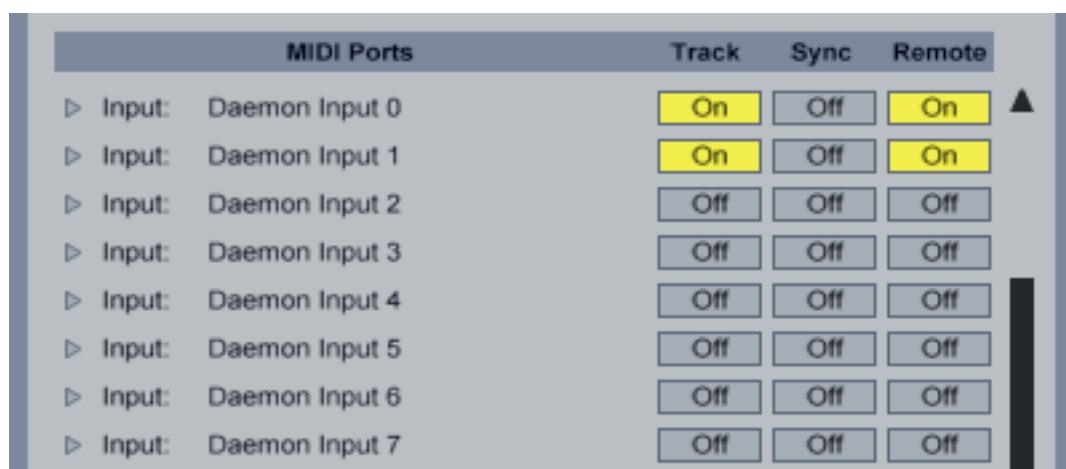
- **Setup MIDI Daemon** opens the MIDI connection browser of the Daemon. Please have a look at Chapter 8 “Targets Setup” for details.
- **Quit** shuts the MIDI Daemon down.

Please note that the Daemon must be running if you want to use MIDI with the Lemur, or if any of your Lemur objects is transmitting mouse or keyboard commands to your computer. Use the small X in the upper left corner of the MIDI port browser for closing it without shutting the Daemon down

If you want the Daemon to be automatically started, enable the **Launch Daemon at startup** checkbox.



On Mac, the JazzDaemon automatically creates 8 virtual MIDI Inputs and 8 virtual MIDI Outputs for you to link with Lemurs. They are conveniently named Daemon Input and Daemon Output. Once connected to the Ins and Outs of your Lemur, you can use them to communicate with any MIDI application on your Mac.

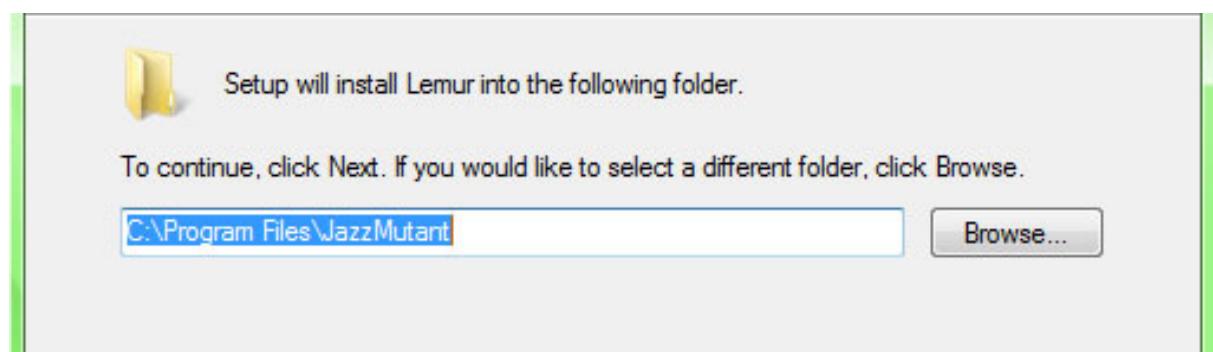


## 4.3 Windows Install

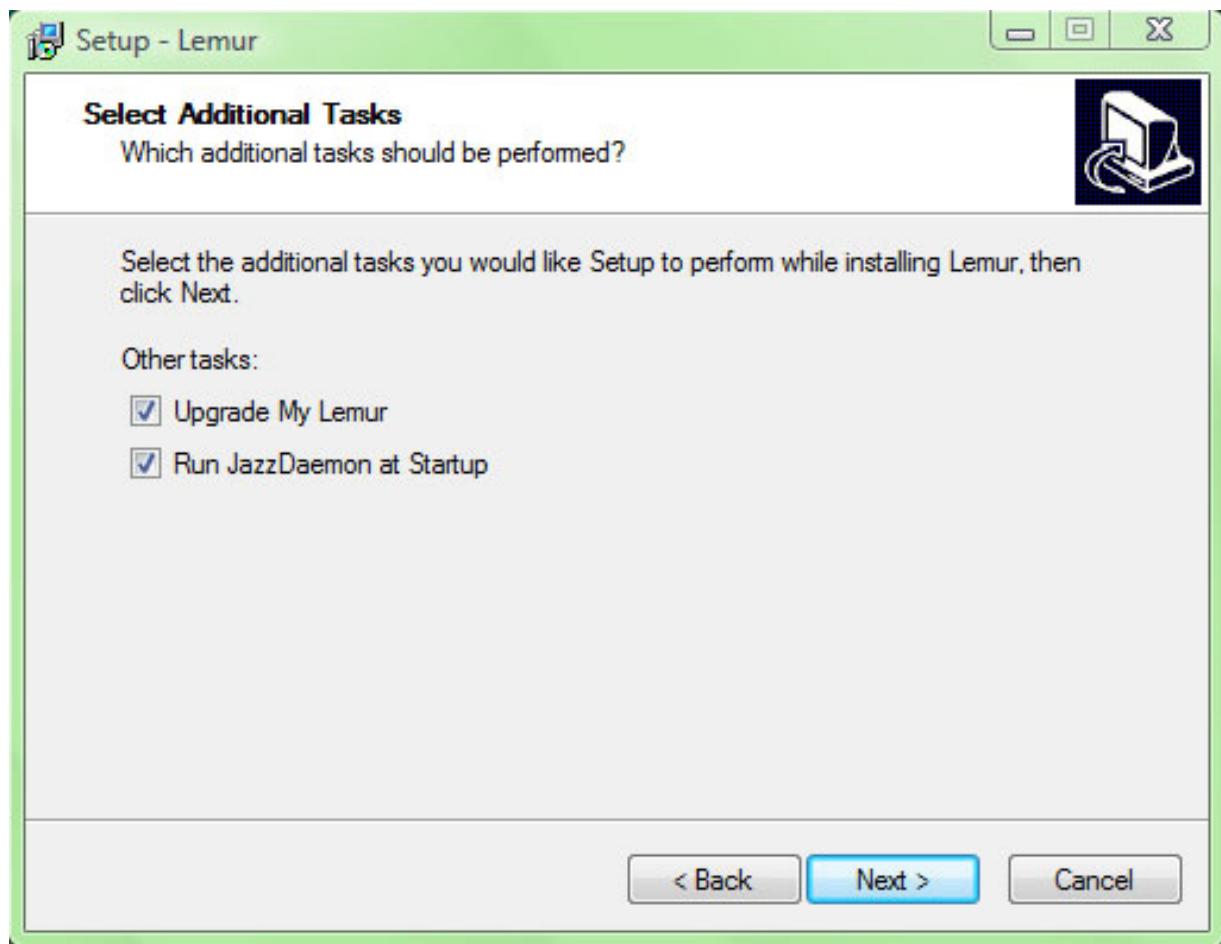
Download the latest software from the web, or insert the installation CD into your PC, double-click the **Lemur.exe** and follow the instructions.



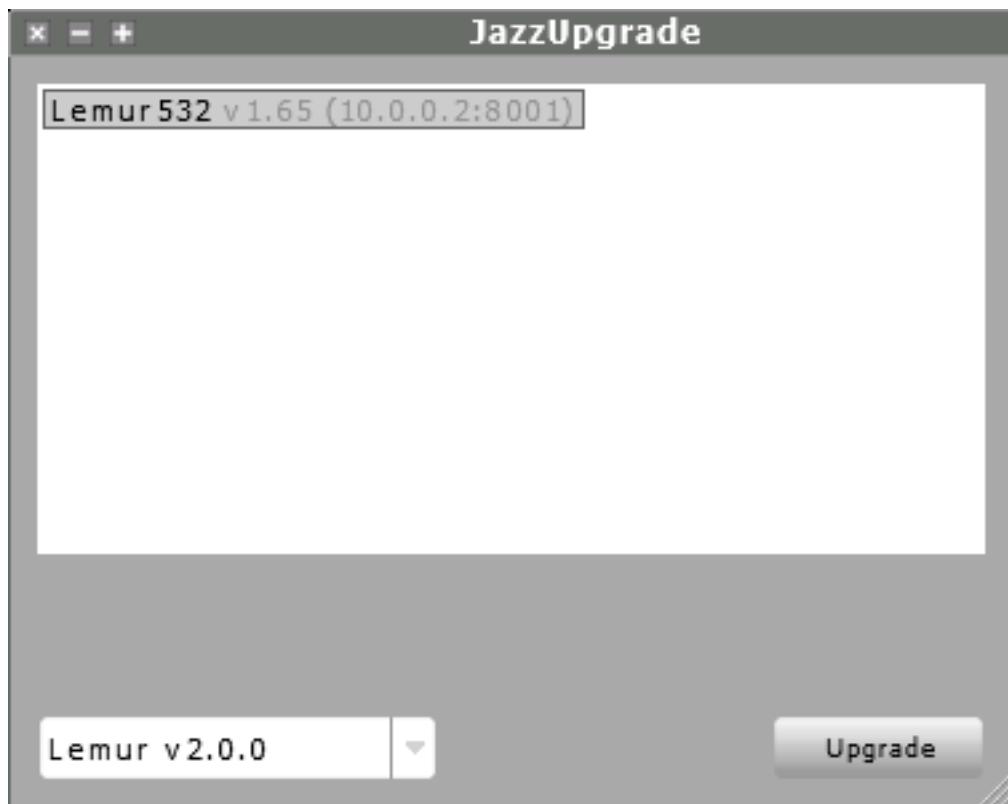
By default the Installer will create a folder named JazzMutant in your Program Files (C:\Program Files\JazzMutant). Click on **Browse** to select a different folder.



If your Lemur runs an older version (the version of the firmware currently installed on your Lemur is indicated on your Lemur's settings page), select the **Upgrade My Lemur** task when prompted to do so.



The JazzUpgrade will then display a window listing the Lemur(s) connected to the network:



Select the Lemur you want to upgrade in the list, and click **Upgrade**. The new firmware will be uploaded to the Lemur, which will then reprogram itself and reboot with the new firmware. If the JazzUpgrade does not list your Lemur in the window, double-check your Network settings, your cable connections, deactivate any firewall you may have running and make sure any previous versions of the JazzEditor or JazzDaemon are turned off.

## 4.4 The JazzDaemon on Windows

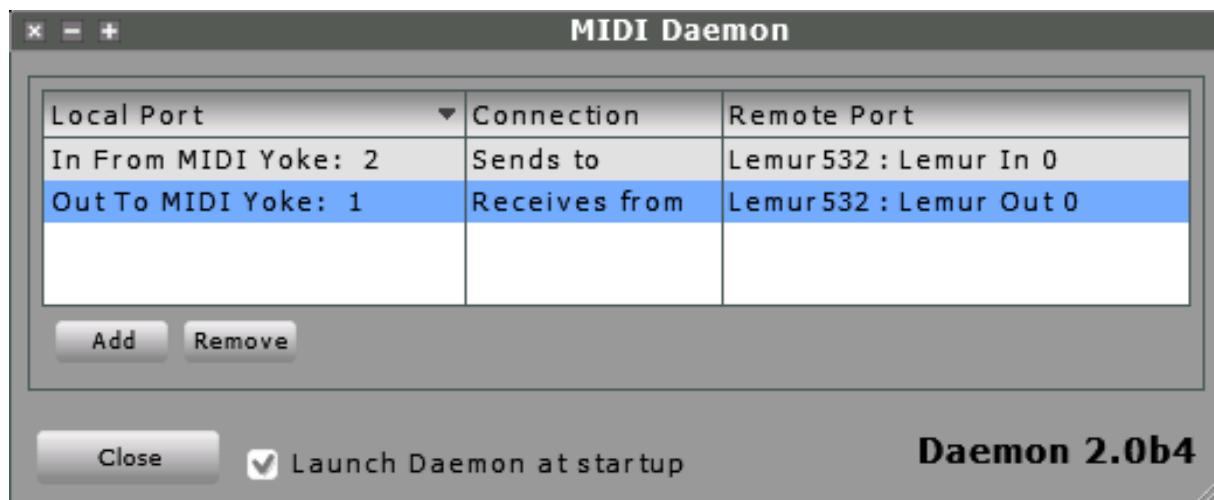
By default the JazzDaemon will be loaded automatically at boot time of the computer and appear in the System Tray on your desktop.



A click on the JazzDaemon icon opens a menu with two entries:

- **Setup MIDI Daemon** opens the MIDI port browser of the Daemon. Please have a look at chapter 8.2 for details.
- **Quit** shuts the MIDI Daemon down.

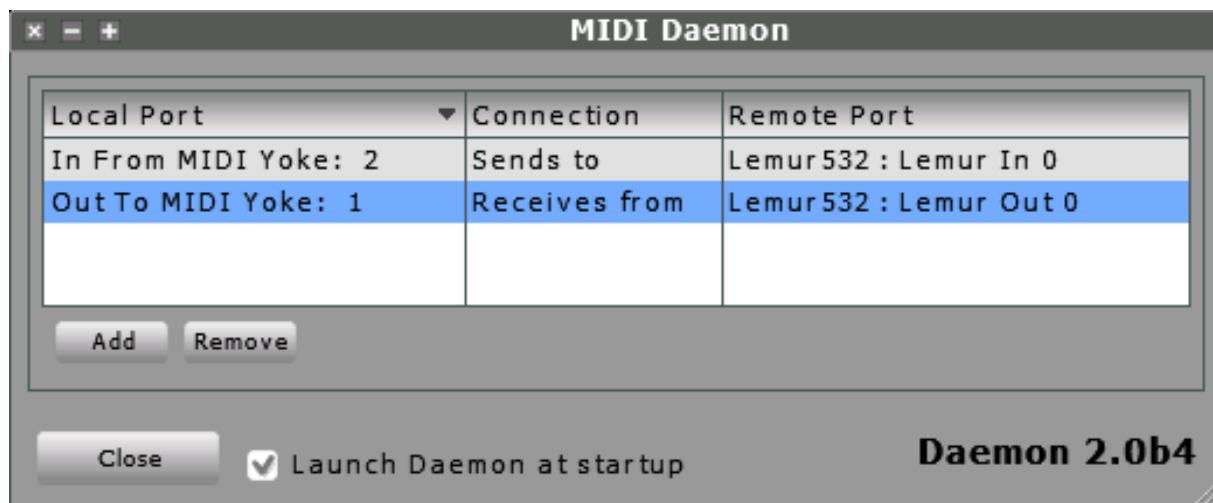
Please note that the JazzDaemon must be running if you want to use MIDI with the Lemur, or if any of your Lemur objects is transmitting mouse or keyboard commands to your computer. Use the small **X** in the upper left corner of the MIDI connection browser for closing it without shutting the Daemon down.



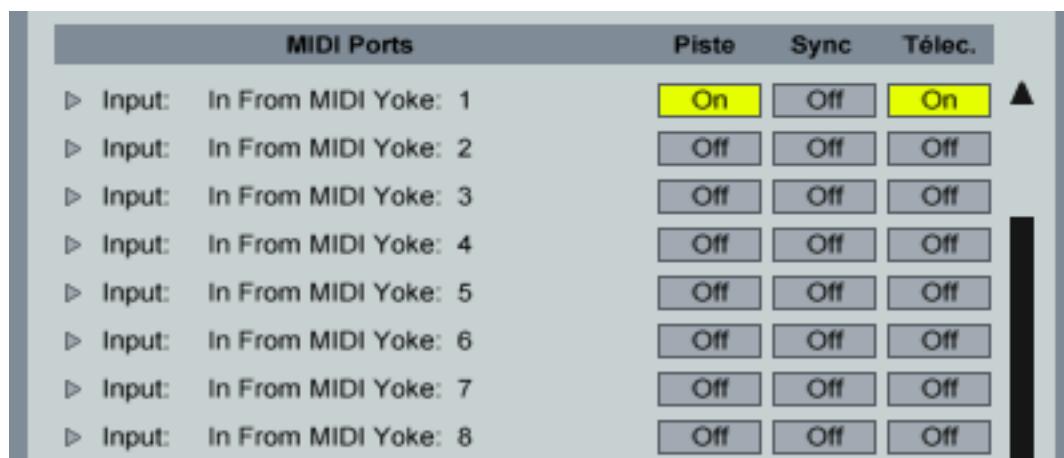
If you don't want the Daemon to be automatically started, you can change its behaviour. Uncheck the **Launch Daemon at Startup** flag and the next time you boot your computer the Daemon won't be started.

On Windows, you will need third-party virtual MIDI ports, such as those created by **Maple MIDI** or **MIDI Yoke**, in order to link your applications to the JazzDaemon. Both drivers are freeware and easily downloadable from the Internet.

In the software application you want to control with the Lemur, choose the virtual MIDI port to which the Lemur is connected. As you can also have bi-directional communication via MIDI - i.e. the MIDI-enabled software can also control Lemur Objects - use a second, distinct virtual port to make the reverse connection to the Lemur. Using a distinct port prevents MIDI feedback problems.



Once you've connected them to the Ins and Outs of Lemurs, you can use them in any MIDI application on your PC.

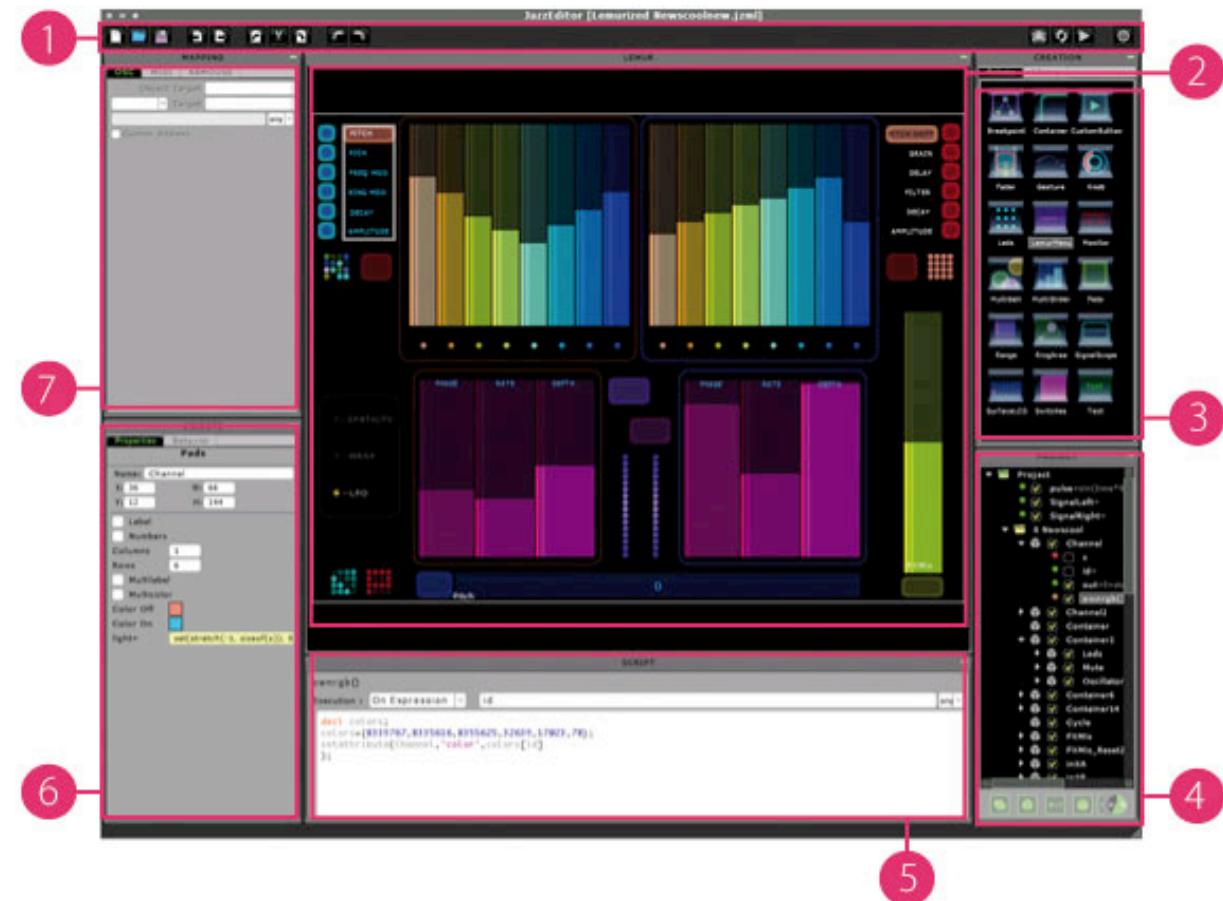


# Chapter 5 - Introducing the JazzEditor

All editing and building of Interfaces is done via the JazzEditor software. Here you create, modify and save your Projects, or simply open them in order to transfer them to your Lemur.

## 5.1 Overview

The JazzEditor's Workspace consists of seven main areas:



**1. Header:** provides icons for basic functions.

**2. Lemur panel:** this is where the virtual objects are placed to build interfaces up.

**3. Creation panel:** contains the Palette and Library tabs.

**4. Project panel:** provides a hierarchical view of your current project.

**5. Script panel:** for defining an expression or a multi-line script.

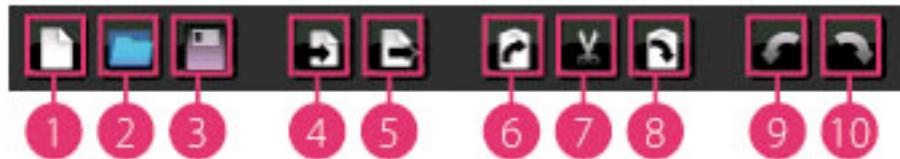
**6. Objects panel:** for defining the selected object's Properties and Behavior.

**7. Mapping panel:** for assigning OSC or MIDI messages to your object's variables.

It is useful to know that the layout of the JazzEditor Workspace can be customized to suit your preferences. Just grab a panel by its title bar, and drag it to another spot on the Workspace. Everything can be rearranged.

## 5.2 Header

The Header section provides a series of icons that allow quick and easy access to some of the JazzEditor's basic functions.



- 1. New Project:** Creates an empty Project.
- 2. Open Project:** Presents you with a dialog for opening Projects from your file system.
- 3. Save Project:** Opens a dialog for saving your current Project to the file system. Choose an appropriate folder to store your Projects.
- 4. Import Module:** Opens a dialog for importing a Module into the currently opened Interface at the level of hierarchy you choose.
- 5. Export Selection:** Exports the selected group of objects as a Module and save it to the file system.
- 6. Copy:** Copies selection to clipboard.
- 7. Cut:** Cuts selection to clipboard.
- 8. Paste:** Pastes from clipboard.
- 9. Undo:** Rolls back an unlimited number of JazzEditor commands.
- 10. Redo:** Redo last command.



- 11. MIDI Mapping:** Opens a list of the current project's MIDI mappings.
- 12. Synchronization:** When the JazzEditor is connected to a Lemur, any changes to Objects' states on the Lemur will also be reflected in the JazzEditor. This can be very handy for saving a Project including the state of all objects.
- 13. Connection:** Opens a dialog listing the accessible Lemurs on the network for connecting the JazzEditor to a Lemur and transferring the current project.
- 14. Settings:** Opens the JazzEditor settings menu.

### 5.3 Lemur Panel

The Lemur panel is the canvas on which your Objects are laid down and assembled to construct Interfaces. In terms of size, it replicates that of your Lemur's screen (800x600 pixels). The Lemur panel always displays the Interface currently selected in the Project panel (see corresponding section below).

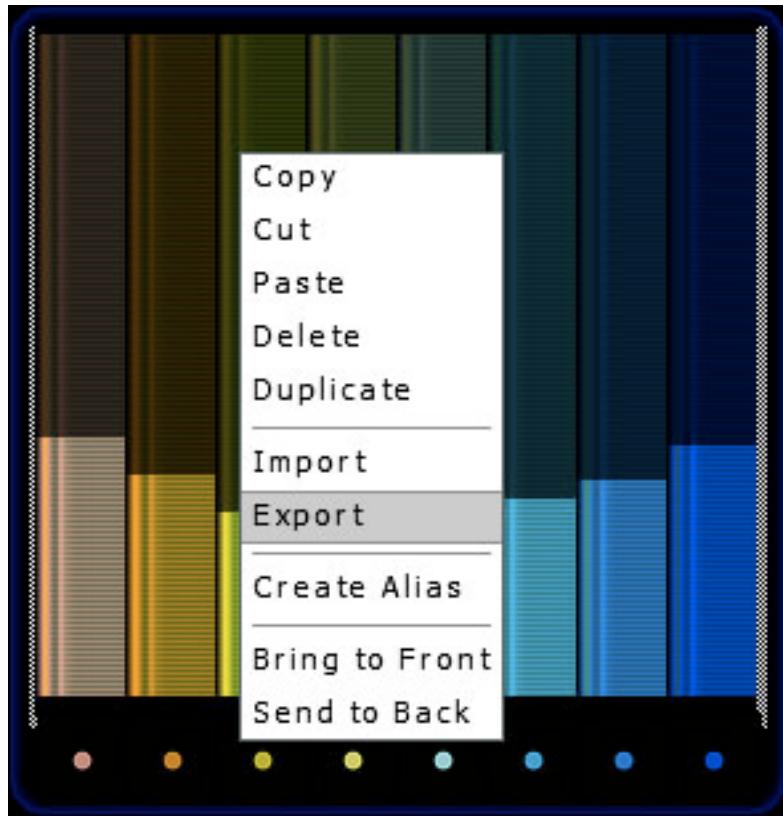


Objects are created either by dragging and dropping the desired icons from the **Palette**, or by clicking on the **Create Object** icon underneath the **Project** panel. Fetch the Objects with your mouse and displace them to the ideal spot. You will notice that when your mouse hovers over an Object, its corners highlight. Simply grab a corner and move the mouse to resize it.

Multiple Objects can be selected by holding the Shift key and clicking on them one after the other. You can also drag a square around them to get the same effect. By clicking an Object again while still holding the Shift key you remove it from the selection.

There are two different modes available concerning interaction with Objects in the JazzEditor. In default **Edit** mode, you can't change the state of the Objects laying on the Lemur panel. In **Run** mode, you can use your mouse to change the state of the Objects, as if you were using your fingers on the Lemur. Simply press "e" on your computer keyboard and hold it to switch to Run mode. As long as you hold the key you can use your Interface. This comes very handy when debugging your work without having your Lemur connected.

The Editing Area also features a **contextual menu** which is displayed by right-clicking (PC)/ ctrl-clicking (Mac) on an Object. It offers an additional way of executing commands such as **Copy** or **Export**:



Objects can overlap on the Interface. The rule of thumb is that the Object created last will cover older Objects. The commands **Bring to Front** or **Send to Back** change the order of Objects in a "pile". Some Objects (like the Monitor) can have a transparent background, leading to various possibilities for labels and captions. When the Transparency flag is activated, only parts of the Object will remain opaque and any Objects lying underneath can be seen.

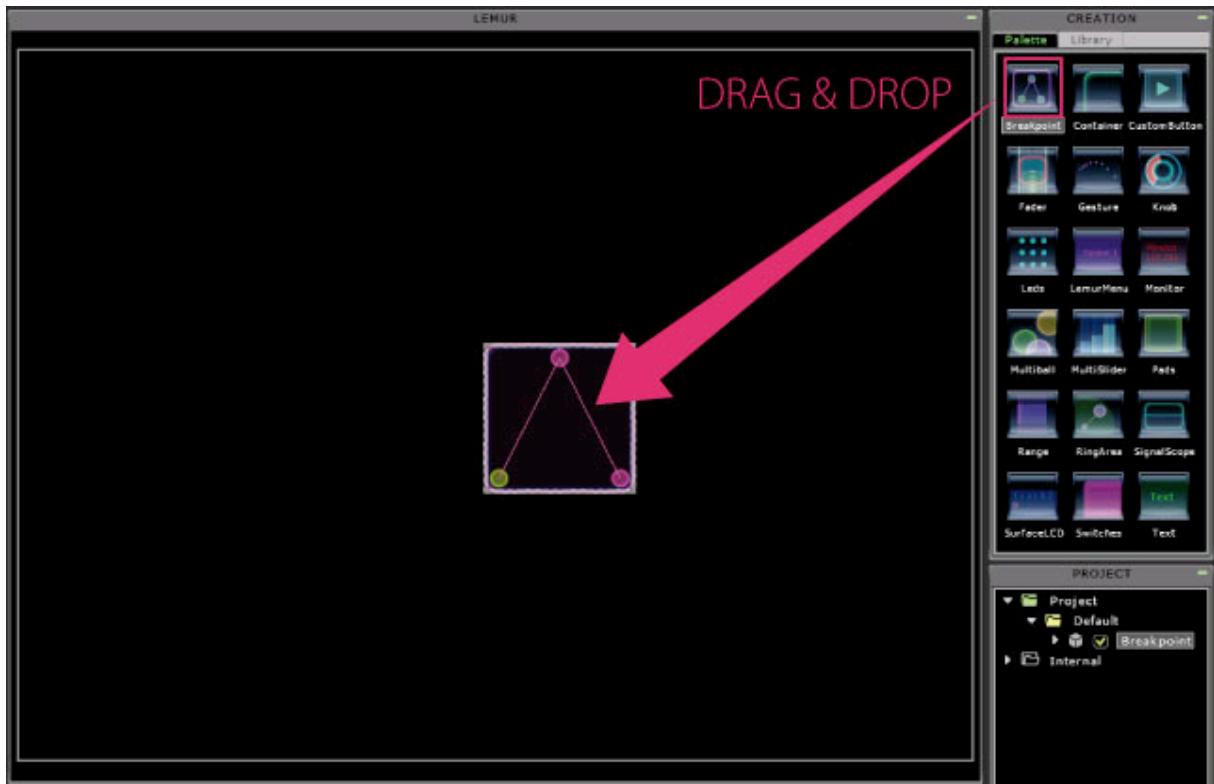
Note that a lot of the commands and functions of the JazzEditor can also be controlled via keyboard **shortcuts**. The shortcuts are displayed in the Tool tips that pop up when your mouse pointer hovers above the respective commands. Please refer to **Appendix IV** for a complete list of shortcuts.

## 5.4 Creation Panel

The Creation panel consists of 2 tabs: the **Palette**, for creating Objects by dragging them from the Objects Icons lists and dropping them on the Lemur panel, and the **Library**, for dragging modules to and from the Lemur panel.

### 5.4.1 Palette

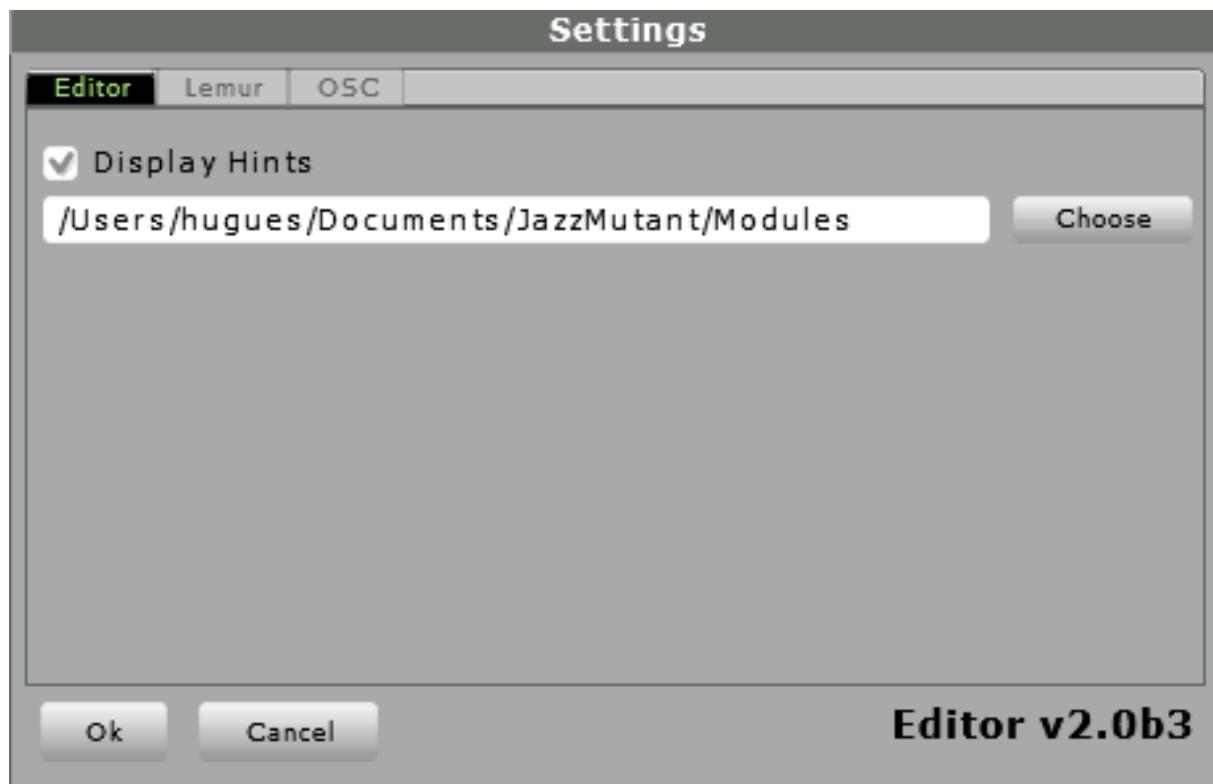
In order to create an Object, select an icon and drag it anywhere on the Lemur panel.



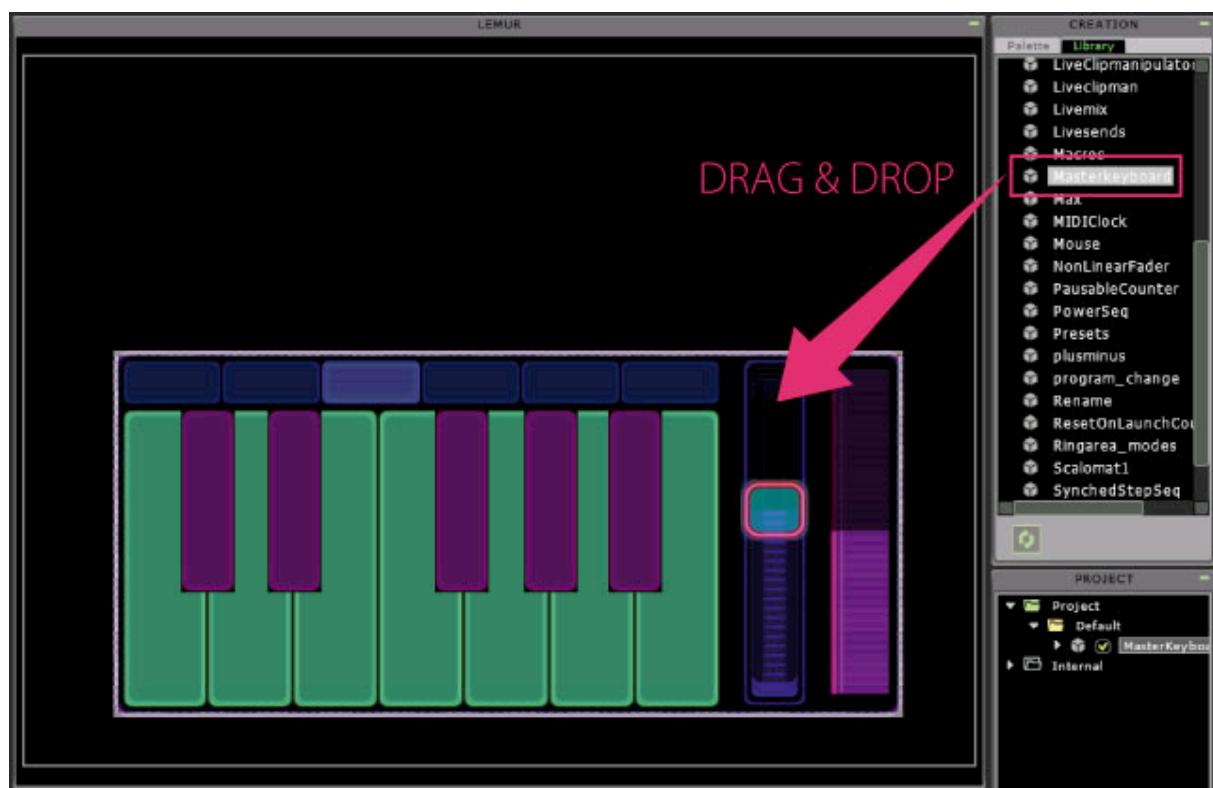
As we'll see later, you can also create an Object by clicking on the **Create Object** icon underneath the **Project** panel. The choice is yours.

### 5.4.2 Library

The **Library** works on the same principle, however before being able to use the Library, a file path for your module folder must be specified in the **JazzEditor Settings** window (click on the top-right corner's icon).



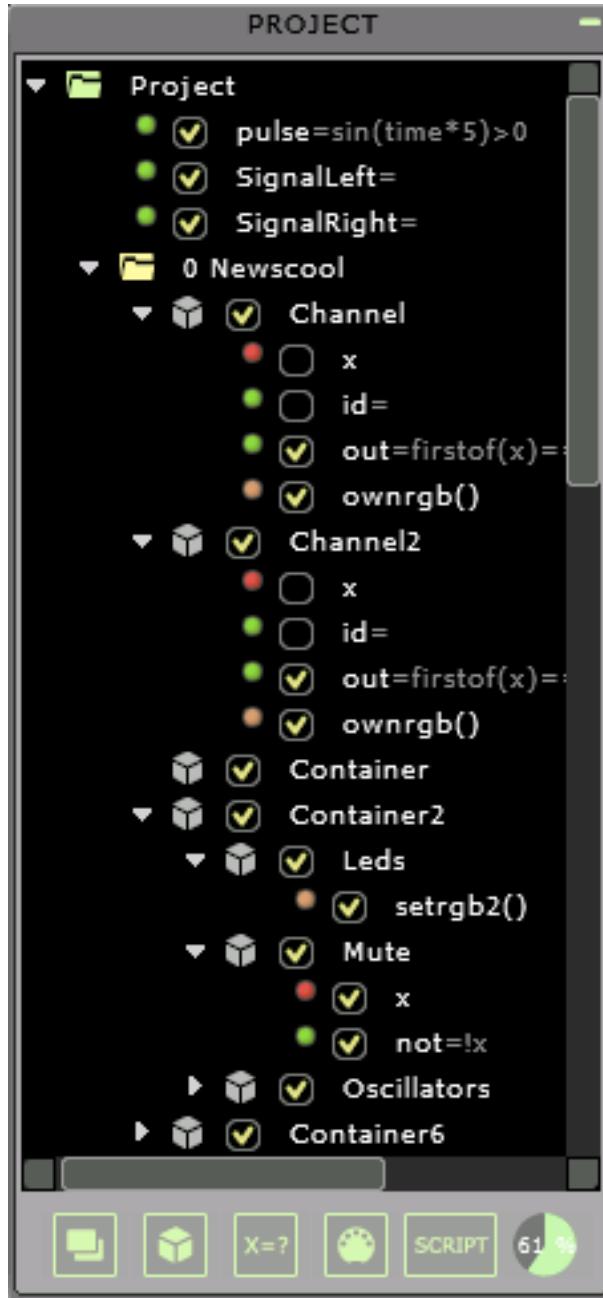
When you're done, the modules present in your folder should be listed and available to be dragged and dropped to the Lemur panel.



This also works the other way around. Any Object selection on the Lemur panel can be dragged to the Library panel by holding the **alt** key and be directly saved as a module.

## 5.5 Project Panel

The Project panel displays all elements of the current Project in a **hierarchical tree** structure.



Click on the small **disclosure triangles** in front of the symbols or double-click on a symbol to expand or collapse the next deeper level of the tree.

A **green folder** on top of the tree structure represents the current Project. **Yellow folders** represent the different Interfaces (pages) contained in the Project. These Interfaces folders contain the Objects, depicted by the small **grey cubes**, together with their built-in Variables, and any additional User defined Variables, Functions, Custom Midi messages or Scripts, all represented by **colored dots**. Note that in some instances it may be necessary to create User defined Variables, Functions, Custom Midi messages or Scripts directly at the root of the Project folder so that they can be used globally, but more will be said on this later.

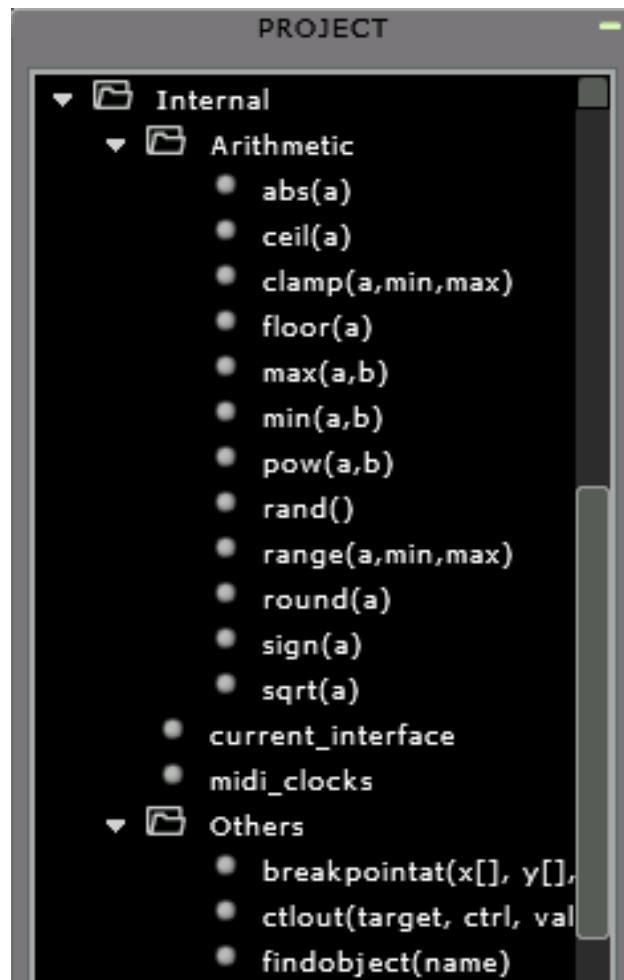
The various entries in the Project Browser are color-coded as follows:

- **Red** dots stand for the Object's main parameters (x for a Fader; x, y, z for a MultiBall, etc.)
- **Orange** dots depict Custom MIDI messages or Scripts created by the User.
- **Green** dots indicate User-defined variables.
- **Blue** dots show User-defined Functions.
- **Grey** depict internal Functions

The small **checkboxes** play an important role in Lemur Projects. If a box is checked, it means that the MIDI or OSC messages associated with the entry, being a single variable or a whole Object, are transmitted to the Targets. In the case of a Script, the checkbox can be used to activate or deactivate the execution of that Script.

There might be at least two reasons for not sending the output value of a particular object. One would be if you were using an Object merely to display information coming from your computer. The MultiBall object could represent the state of something happening in an application for instance.

The other reason could be that you transmit the data from the Object to your computer via a User-defined Variable containing a mathematical expression. In this case the built-in Variable (e.g. the x) is only referenced by the Expression in another Variable. This is often used to scale values into a more appropriate range. We'll learn more about this technique later.



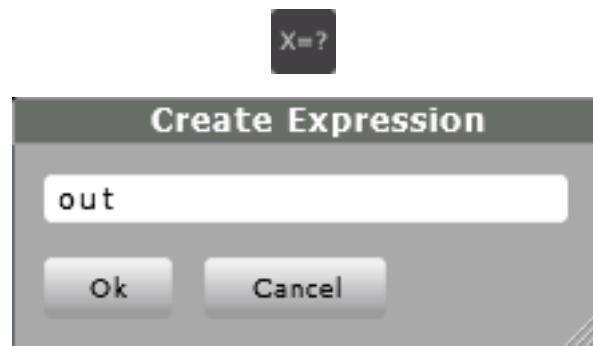
Use the five **Create** commands to create new Interfaces, Objects, Expressions, Custom MIDI mappings, and Scripts.



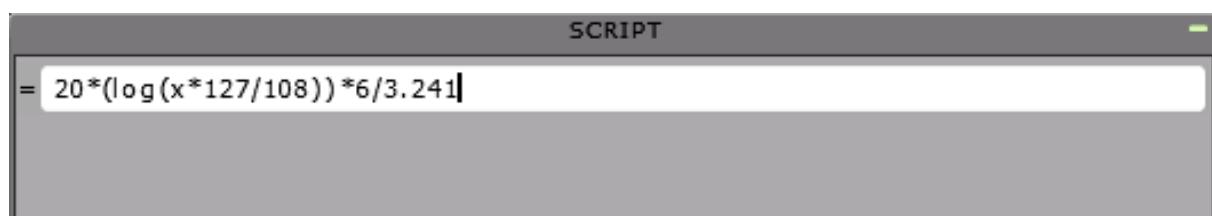
To the right, the **Memory Display** shows you the percentage of the Lemur's memory that your Project will occupy. This is for reference only.

## 5.6 Script Panel

The Script panel is multi-purpose. It allows for the single-line definition of **User-defined Variables** or **Functions** created with the **Create Expression** icon underneath the Project Browser. For instance, imagine we need to derive a logarithmic value from a Fader's built-in x Variable. Click on the Create Expression icon and name it appropriately:

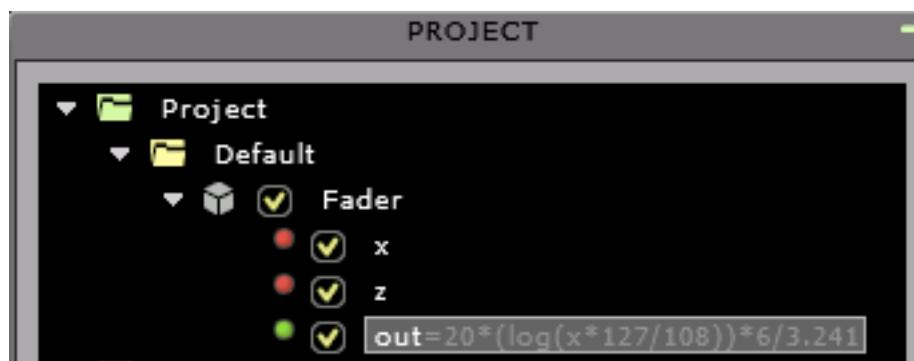


Then type in the mathematical expression to define your own Variable with respect to the original x variable:

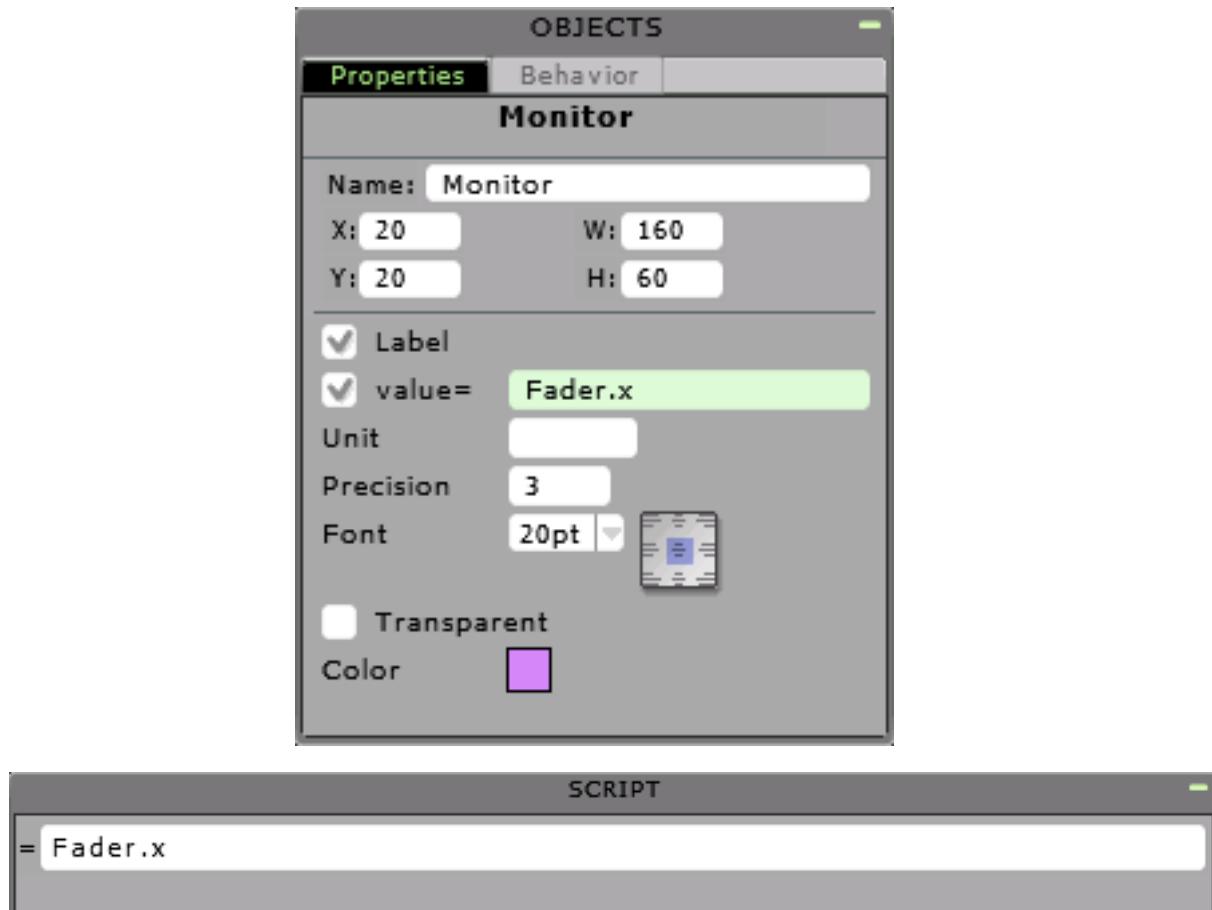


Notice here that we've used a **log** function. This is one of the many handy built-in functions, available to make your life easier when building complex interfaces. Please refer to the Parser reference on Chapter 13 for insights into built-in functions and operators.

This User-defined Variable we just created is then displayed in the Project panel and ready to be used.

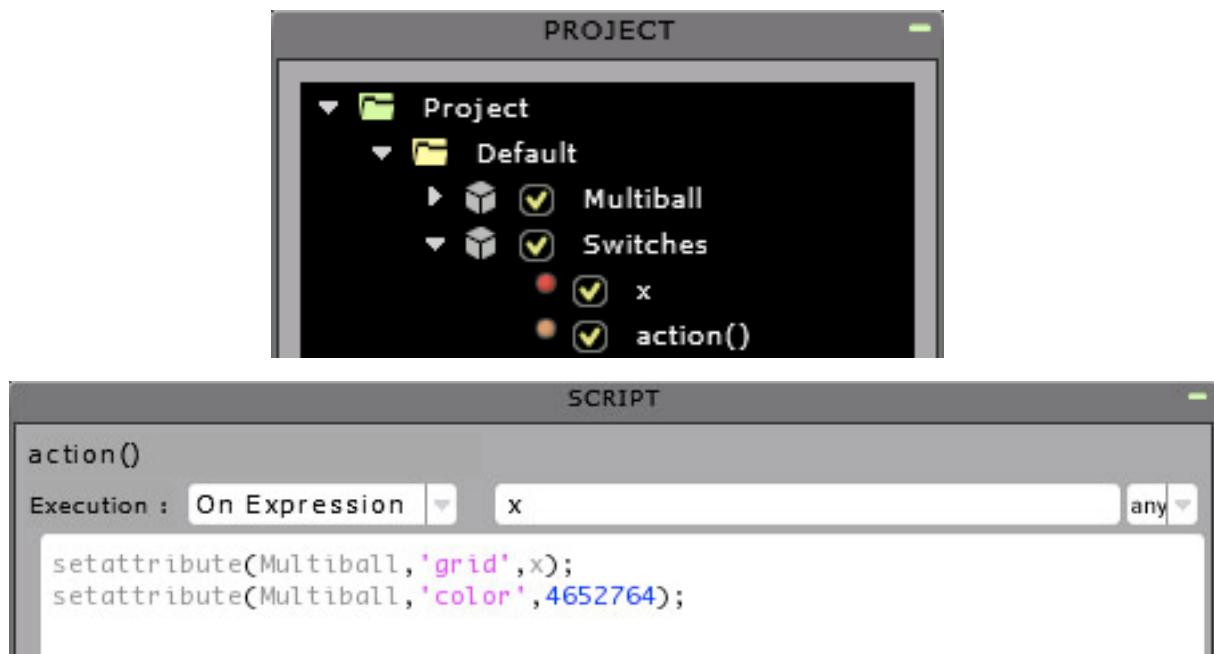


The Script panel can also be used to enter a single-line expression for some User-defined **Object parameters**, typically the Light or Value parameters:



Finally, the Script panel also allows for the **multi-line** coding of a **Script** created with the **Create Script** icon. Multi-line scripts can be used, in conjunction with a set of powerful **built-in functions**, to instruct your Lemur to perform specific tasks at specific times. A typical application of this powerful feature could be the real-time manipulation of an object' **attributes**, such as its dimensions, color or physics. A basic example would be to instruct a Switch to activate a Multiball's grid, or a Text object to change its content upon reception of an OSC or MIDI message.





Please refer to Chapter 11 and 12 for more information about multi-line scripts.

## 5.7 Objects Panel

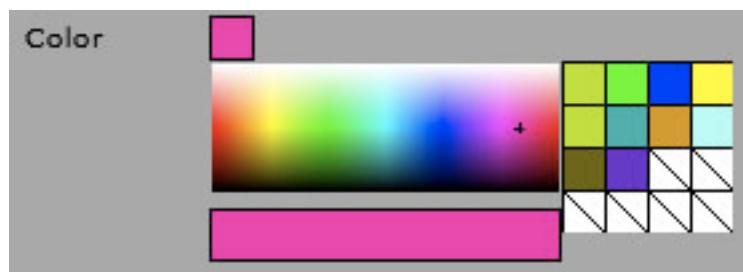
The Objects panel contains two tabs: the **Properties** and **Behavior** tabs. The Properties tab provides access to the basic properties of the Lemur's Objects, while the Behavior tab concerns their physical responses. Details of all the Objects's parameters can be found in the Object Reference section of this manual.

### 5.7.1 Properties



In essence, the Properties tab deals with the appearance of the Objects such as their color for instance. This is also where you name an Object (up to 64 characters) or type in its size and position with greater precision if you need to. Additional parameters may vary depending on the type of Object selected.

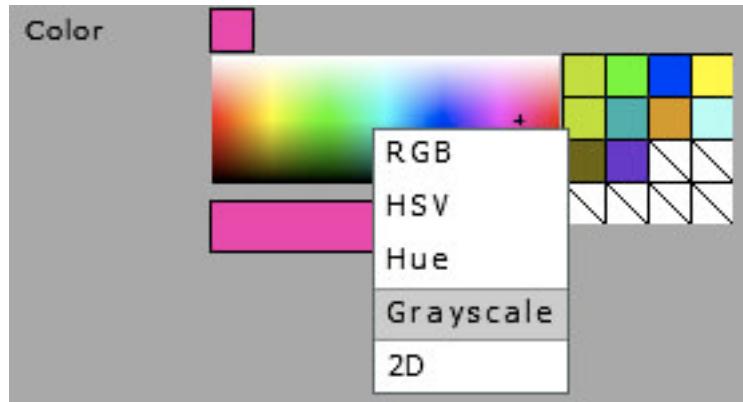
By default, the **Color** parameter of your Objects is set through a 2D color picker, which appears when you click on the square color sample:



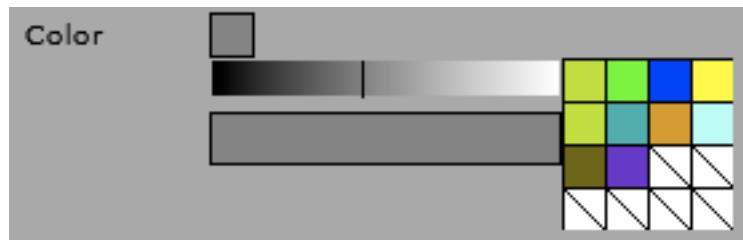
The grid to the right of the color picker is for storing your favorite colors. Just drag the large color sample bar underneath the color picker to a slot in the grid, and that color will be available for direct selection in the grid next time you use the color picker.

Note that the small square color sample next to **Color** can be directly dragged to other Objects present on the Lemur panel to colour them with the exact same colour.

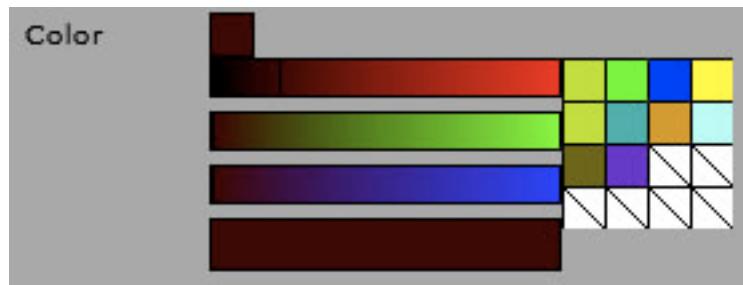
If needed, the colour selection mode system can be changed by right-clicking (PC) or ctrl-clicking (Mac) on the colour picker:



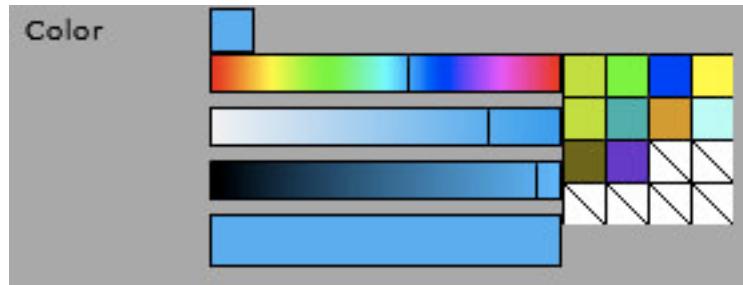
Choosing **Grayscale** for instance will leave you with a black and white selection tool.



Choosing **RGB** leaves you with a classical RGB additive color selection tool, in which the red, green and blue sliders are added together to produce the desired color.



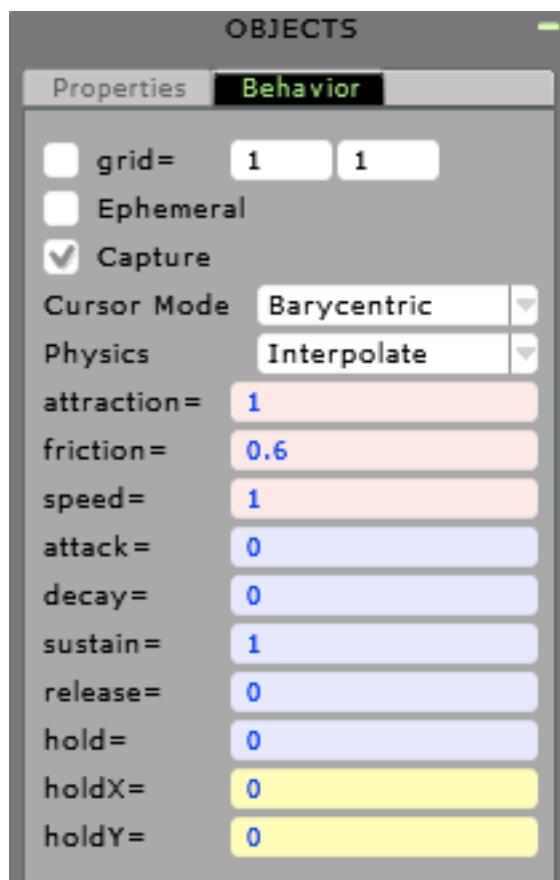
**HSV** stands for hue, saturation, value, where the hue slider lets you pick up the “pure” color, saturation the perceived “intensity” and value the “lightness”.



Choosing **Hue** leaves you with the traditional, simple color swatch tool:



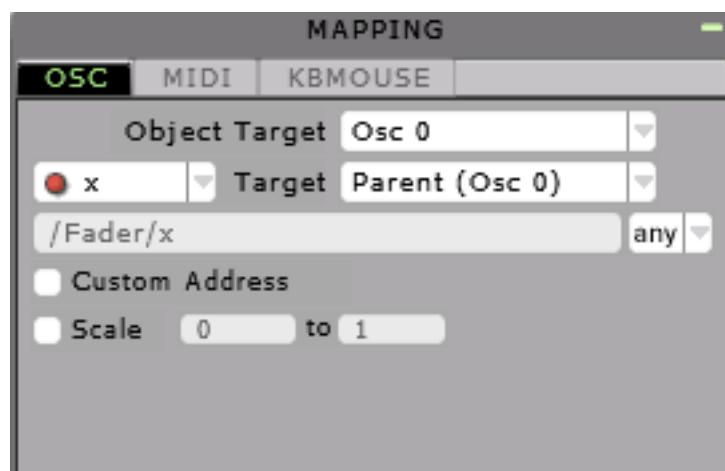
## 5.7.2 Behavior



The Behavior tab circles around the physical characteristics of the Objects. Here you find parameters like friction and tension and the different physics modes of the Object.

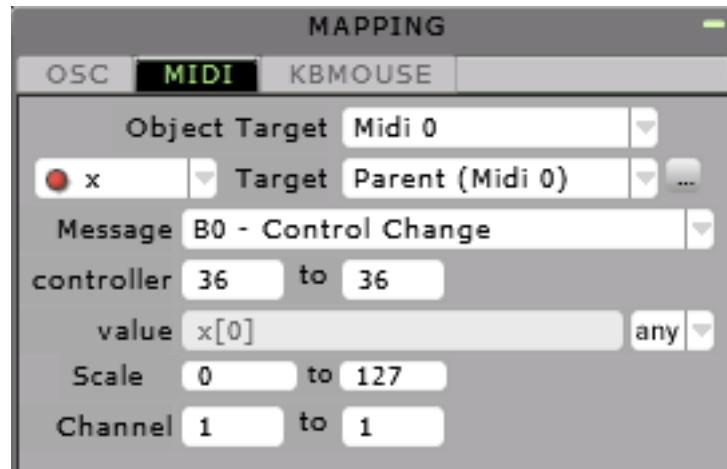
## 5.8 Mapping Panel

### 5.8.1 OSC



On the OSC Panel you can define the routing of the different Variables to OSC **Targets**. You have a menu for the Variables and one for the eight possible OSC Targets. This panel also serves for controlling Lemur objects via OSC.

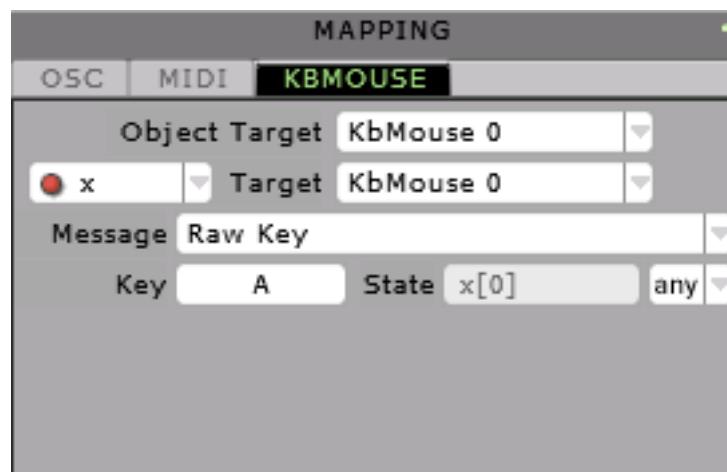
## 5.8.2 MIDI



The MIDI panel lets you assign **MIDI messages** to your Variables and route them to the MIDI Targets. This panel also serve for controlling Lemur Objects via MIDI.

## 5.8.3 Keyboard and Mouse

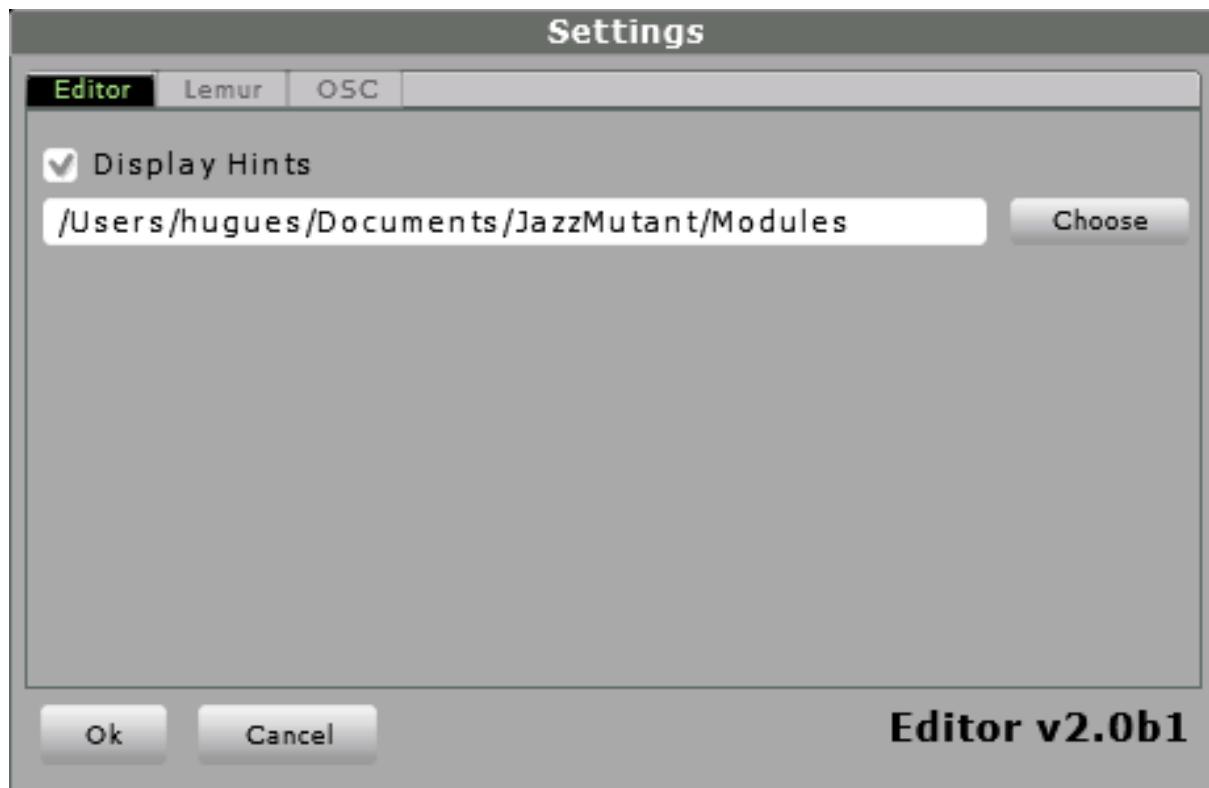
The KBMOUSE panel is for assigning **key commands**, **mouse movements** and **mouse clicks** to your Object's variables.



## 5.9 Settings Menu

### 5.9.1 Editor

The first panel of the Editor tab deals with two basic settings. The first is the **Display Hints** checkbox: when active, the description of the interface item alongside with the eventual keyboard shortcut is displayed when you hover with your mouse pointer over the various interface elements.



The second is the path selection for the Creation panel's **Library**. Here you tell the JazzEditor where to look for your Modules collection, so that afterwards you may simply drag and drop them to the Lemur panel.

## 5.9.2 Lemur

The second panel of the Settings window, called Lemur, deals with two other settings. The first is the **Automap** checkbox: when active, each new Object will be automatically mapped to the chosen MIDI Target and Message:

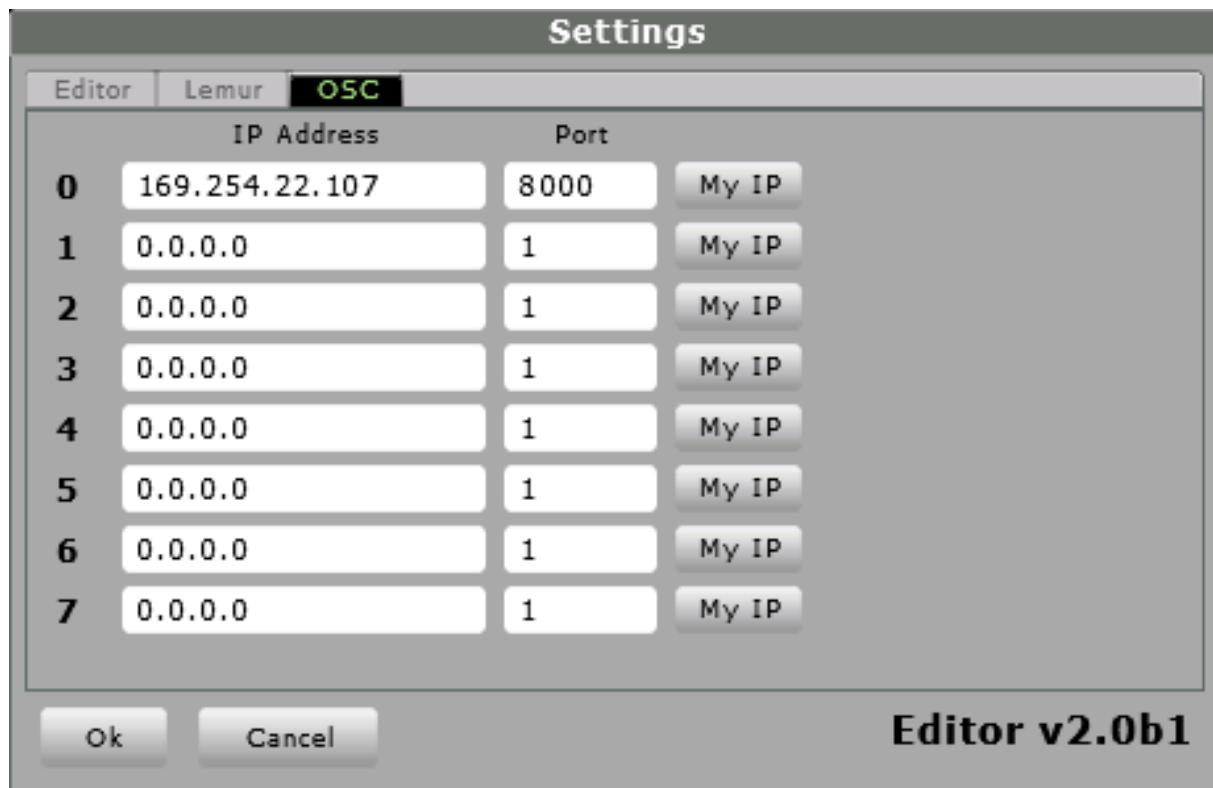


The **Local IP** setting lets you choose which of your computer's local IP addresses the JazzEditor will use for connecting to the Lemur.

The IP of the currently connected Lemur is also displayed.

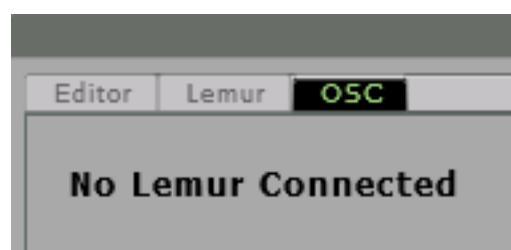
### 5.9.3 OSC

The third panel of the Settings window lets you set your **OSC Targets** provided the JazzEditor is connected to your Lemur. This panel mirrors the OSC Targets settings page on your Lemur, and either can be used to configure your OSC Targets.



Just fill in the IP addresses of the Targets you want to communicate with. If it's a piece of software running on the same computer as the JazzEditor, click the **My IP** button, this will automatically fill in the IP of the currently used interface.

If there is no connection, the JazzEditor states this and does not show any OSC settings:



# Chapter 6 - First Steps

## 6.1 Connecting your Lemur

The first step naturally consists in opening the connection between JazzEditor and the Lemur. To do so, click on the **Connect** button on the toolbar, which should result in opening the following window:



Provided your network is set up correctly (see Chapter 3 and Appendix I), the connected Lemurs should be listed with their IP addresses and port numbers. You can select a Lemur on the list and click Connect to establish a connection. A double-click on the list entry does the same trick.

If you just connected your Lemur or had it switched on shortly before, wait a few seconds for the list to update.

Note : Be careful! If you have a Project residing on your Lemur and connect to it with a different Project open in the JazzEditor, the one on the Lemur will be overwritten. A connection always automatically transfers the currently opened Project to the Lemur.

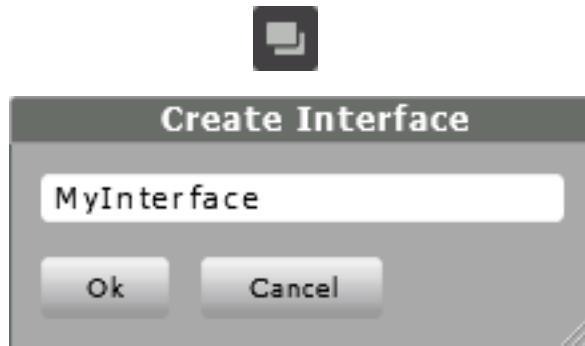
The **IP field** below the list allow manual entry and remembers the last setting of the Lemur connection. It can also be handy for connecting through a proxy.



The current connection status can be seen from the displayed Connection icon. A **Play** button means there is no Lemur currently connected. If you connect to a Lemur the **Stop** button appears and a click leads to a disconnection of the Lemur.

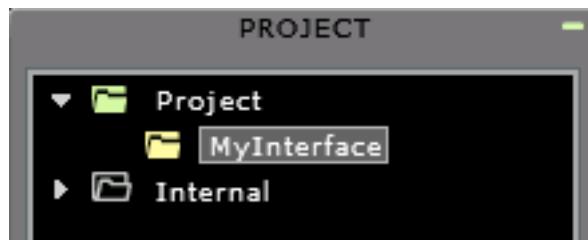
## 6.2 Creating an Interface

Now that the Lemur and JazzEditor are connected together, let's create our very first interface. To do so, just click on the **New Interface** button, located below the **Project** panel.

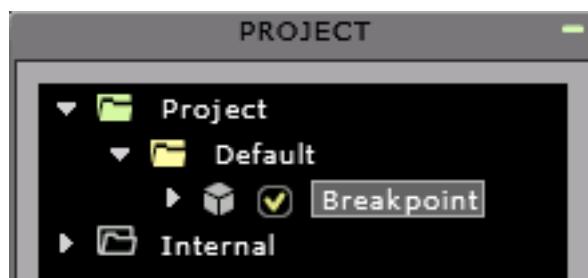


This opens a small dialog asking for the name of the new Interface. You can type in any name you like. Click **OK** and your new Interface is created.

Both the Lemur panel and the Lemur screen suddenly turn black. There is nothing to worry about: we just created a blank interface. Notice that the new interface appears as a **yellow** folder in the Project panel, as shown below.

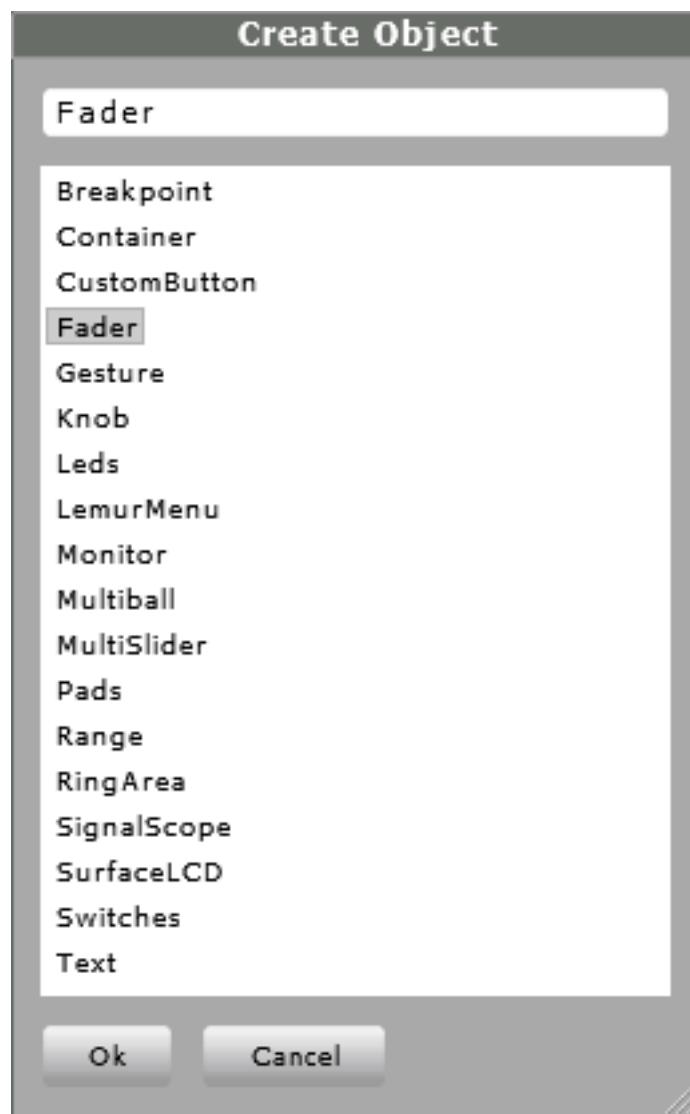


Alternatively, beginning a new project simply by creating / dropping an Object on the Lemur panel will automatically create an Interface named "Default".



## 6.3 Creating Objects

Now to start filling this empty space, you can either use the Palette and drag and drop the desired Object onto the Lemur panel, or click on the **Create Object** button located at the bottom of the Project panel. Clicking on the Create Object button opens the Create Object window, listing the available Object types by alphabetical order:



To create a new object, we just have to choose one among the list, to give it a name of our liking then to click the **Ok** button. For our first experience with interface building, I would suggest to start with the most common: the Fader. Once created, our newbie Fader appears on the upper left corner of both the editing window and the Lemur's screen. It's also referenced in the Project panel.



When you touch the Fader on the Lemur, you will notice that the actions are not mirrored in JazzEditor.



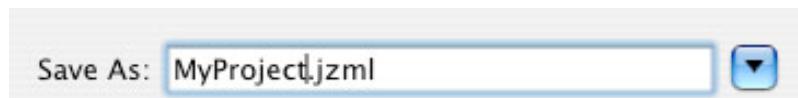
Click the **Synchronization** button to change this. Now everything you do on the Lemur will be reflected on your computer. This helps if you have to save a Project in a defined state for future use.

Already bored with your lonely fader? That's no problem; let's repeat the very same procedure described above and create a Switch and a MultiBall Object.

Not surprisingly, these will appear on both the editing window and the Lemur screen. And as good news never come alone, the two objects are also listed in the Project panel, sorted in alphabetical order.

## 6.4 Saving your Project

You might want to keep this work of art ready for future loading and editing. Just hit the **Save** button in the toolbar and give a name to your project.



All files the JazzEditor produces, be it Project files or Modules, reside on your hard disk, Project files with the extension **.jzml** and Modules with the extension **.jzlib**.



**MyModule.jzlib**



**MyProject.jzml**

Now, let's open our project again, by hitting the **Open** button. You might have noticed that the **Connect** Button has changed its status. That's normal behaviour. When opening a new Project in JazzEditor (either by loading an existing one from disk or by creating an empty Project), the connection to the Lemur is dropped.

If you connect the Lemur again, the Project loaded in JazzEditor will automatically be transferred to the Lemur, overwriting the one displayed on the Lemur.

Make sure to save your work in the JazzEditor before closing the Project.

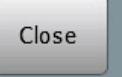
Once your Project is transferred to the Lemur, you may close the JazzEditor and use the Lemur to produce OSC data. It will happily connect to all defined OSC Targets. Remember that if you use MIDI in your Project, the **JazzDaemon** has to run as it is needed to translate the OSC data from the Lemur into MIDI messages.

## 6.5 The Lemur's memory for Projects

The Lemur has a memory for the Projects you throw at it. You just have to tell it to remember them. This frees you from always having to open the JazzEditor when you just want to work with your finished Interface.

Push the second button on top of the Lemur and hit the **Project Browser** button to open the Project browser. It displays a list of Projects currently stored in the Lemur's flash memory.

FM8	
Name	Size
Aerobic	131 Kb
DjKabuki	186 Kb
FM8	109 Kb
FMSynthesis	12.7 Kb
FourtrackLive	39.3 Kb
GoBox	62.7 Kb
GranularSynthesis	14.4 Kb
Mainstage	41.9 Kb
Massive	163 Kb
MCLive	123 Kb
Monemur	20.7 Kb
modul8	118 Kb
Skrewell Lemurized	72.8 Kb
StepSeq	42.3 Kb
SuperSampler	32.6 Kb
testvienna	4.26 Kb
Vectomy	93.2 Kb

We want to save the important Interface we have built, so we touch the **Save Current** command. You see the Project appear on the list. You can select the Project ((leading to a blue tag) and touch **Delete** to remove it from memory and **Open** to load the selected Project into the main memory. You have a maximum of 2048 kb for storing Project data on the unit.

If you want the Lemur to open a specific Project when the Lemur is switched on, select the Project on the list and activate the **Load at Startup** button. The Startup Project is tagged in red. Touch Load at Startup again to remove the tag.

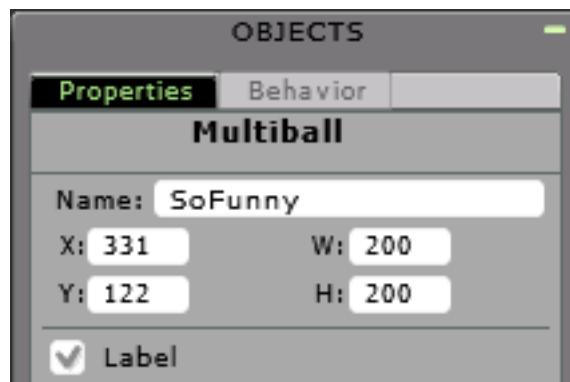
Use the **Close** button to shut down the Project browser and get back to working on the JazzEditor. You can also hit the **Interface Browser** button to go back to the Interface thumbnails.

## 6.6 Changing Object Appearance

First we are going to resize the Fader to have more space for our fingers. Grab the lower right corner of the Fader and drag it down to make the Fader longer and maybe a bit wider too. Note how the Width and Height Parameters in the Properties change as you do it, and vice versa: if you type a value in one of the dimension fields, the fader will move and resize accordingly.

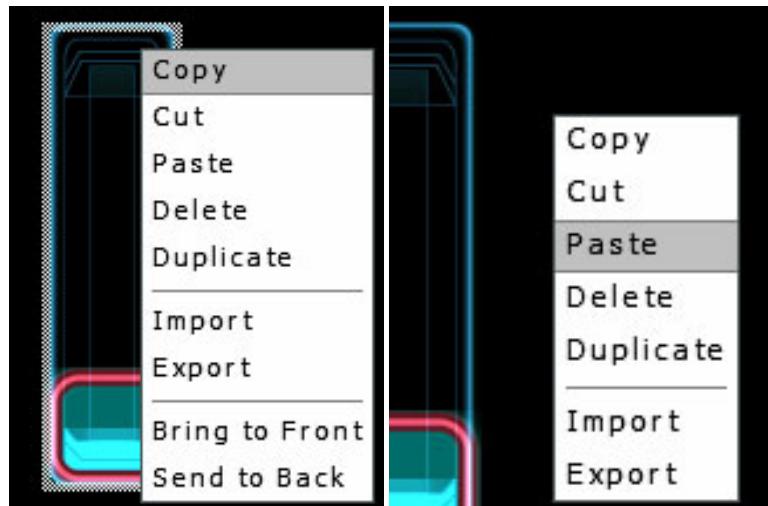
Talking about the Properties tab, what about changing the colour of our fader? Sure, a yellowish green can be nice, but what about a deep blue?

Sometimes it's a good idea to display the name of your Objects on the Interface. Click into the Name field of the MultiBall's Properties tab and change it. Then check the **Label** checkbox.



## 6.7 Groups

Now we want two Faders. We can do this with a simple **Copy** and **Paste**.

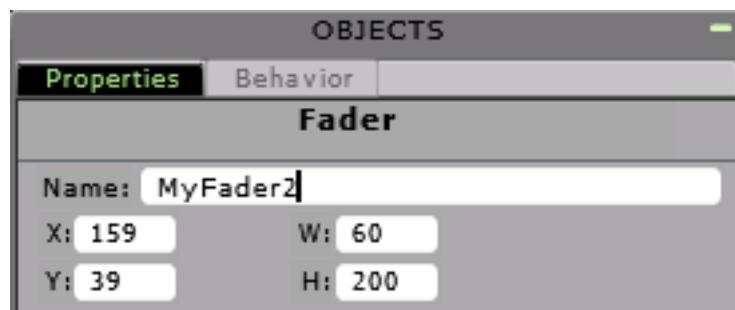


Right-click (Command-click for Mac) on the Fader and choose **Copy** from the context menu. Then right-click on the background of the Lemur panel and choose **Paste** from the menu. Those commands are also available on the Toolbar, via shortcuts, or as an alternative, by holding the **alt** key and dragging the mouse to the desired location.

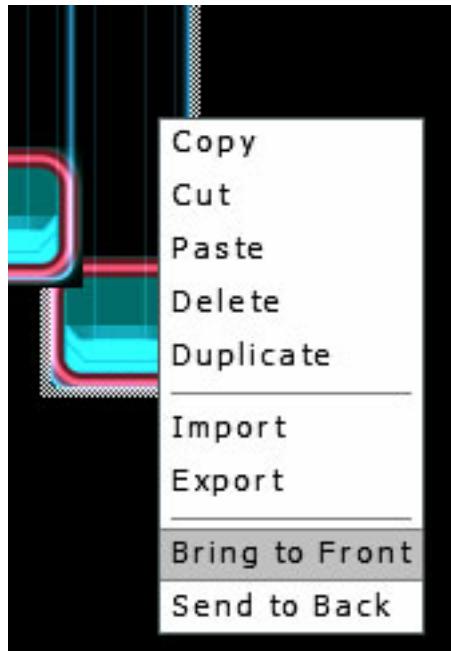
Your Fader is duplicated. Grab it with your mouse and position it to the right of the first Fader.

Have a look at the **Properties** tabs of your Faders. The JazzEditor automatically named the new Fader "MyFader2" to avoid a name conflict. This is important, as the names are also used as addresses of the Objects for OSC mapping and variable accessing. As a consequence, there cannot be any identical names in any Lemur Project – except by protecting objects inside containers, which we'll cover later in this manual.

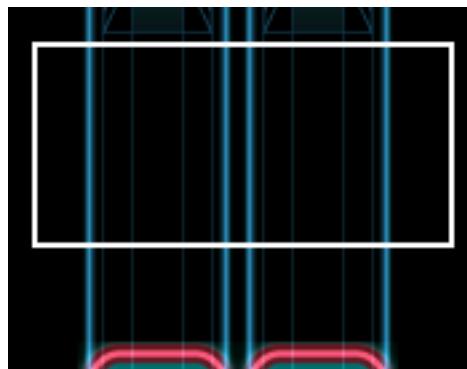
Of course you can also change the names by entering one of your choice into the **Name** field of the Properties.



If you have several Objects lying on top of each other, the JazzEditor provides a way to control the layering order. A right-click on a selected Object opens a menu. Choose **Bring to Front** to make the selected Object the topmost. **Send to Back** puts it to the bottom of the Object pile.



We now want to change the colour of the two Faders. As we already saw, we could do it by changing every Fader on its own. But it's tough to hit the exact same colour twice, so we'll select them both and simultaneously change their colours by dragging a square that touches both Objects with the mouse.

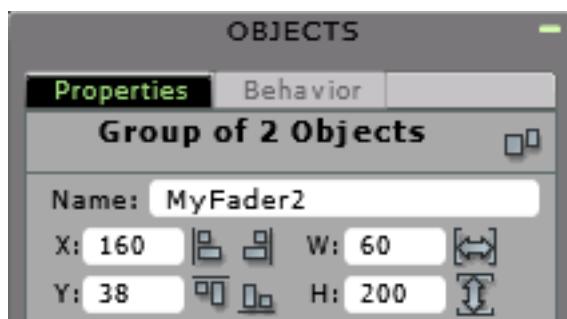


Objects can also be grouped permanently via the context menu by right-clicking (command-clicking for Macs) on the selected Objects and choosing **Group**.



This group command is also available on the Properties panel if several Objects are selected. Do this for your Faders and every time you click on one of them both will be selected. Double-click on your new Group to display the Properties panel. Note that since different kind of objects can grouped together, only the properties common to all objects are displayed. Now dial in a nice purple with the color picker. Both Faders now have the exact same color. If you want to ungroup the Faders, click on the Ungroup symbol in the properties panel.

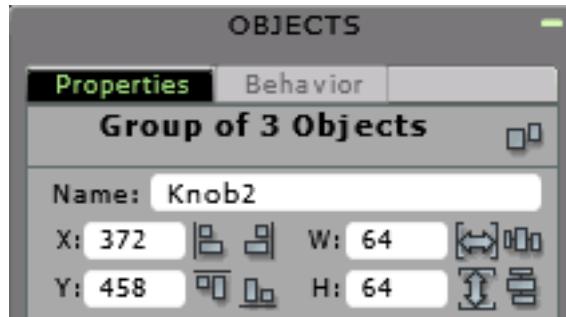
When you select several Objects or have them grouped, you'll notice additional commands appearing on the Properties panel that deal with layout and size of the Objects.



The commands are:

- Align to Left, Align to Right
- Align to Top, Align to Bottom
- Make same Width
- Make same Height

They always refer to the Object you selected first in the group. These commands are of great help when it comes to tidying up your Interfaces. Experiment a bit with those to get the hang, and remember that the **Undo** command is at your disposal to roll back changes.

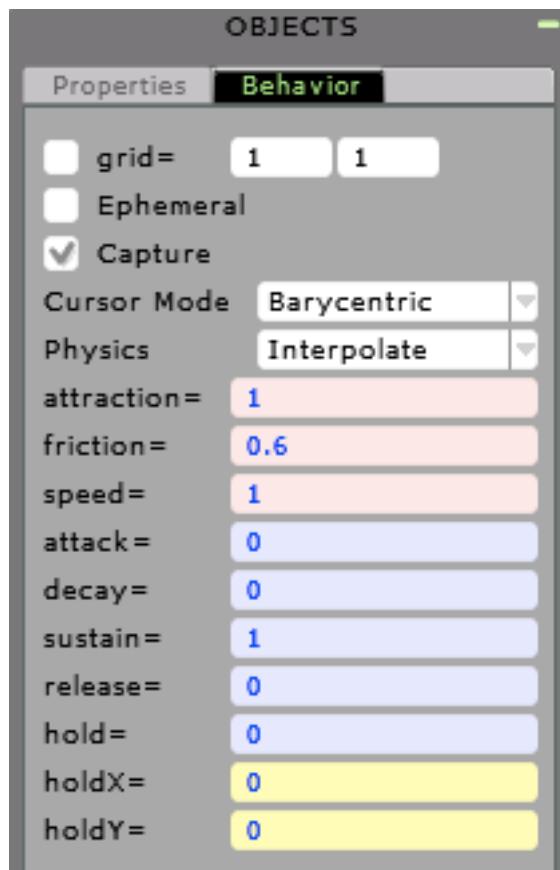


If you select three or more Objects (or have a group of three or more), two additional commands are available:

- **Distribute horizontally** evenly distributes the Objects on the screen. The left- and rightmost Objects are the boundaries for the line of Objects.
- **Distribute vertically** does the same, only on the vertical axis.

## 6.8 Configuring Object's Behaviors

Now, let's glance at one of the most powerful feature of the Lemur's objects: their configurable behaviors. Select the MultiBall object with the mouse, and look at its **Behavior** tab:



The **Physics** menu has three modes defining the behavior of the balls. When Physics is set to **None**, the ball moves immediately to the position of your finger and stays there. It also follows your finger around as you move it on the touch screen, and immediately stops when you lift up your finger.

Make sure the Physics menu is set to **Interpolate**, which should be the default setting.

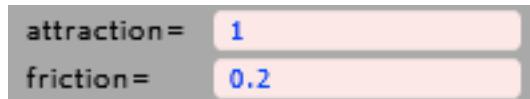


Touch the Lemur screen somewhere in the MultiBall, but not on top of a ball. The ball will not move to your finger immediately but according to the setting of the **Attraction** parameter. Let's modify Attraction to make the ball react more slowly. Enter a value of 0.1. Experiment again with the Ball's reaction. It should follow your finger slowly.

Now let's try the last behavior mode. Choose **Mass-Spring** from the **Physics** menu.



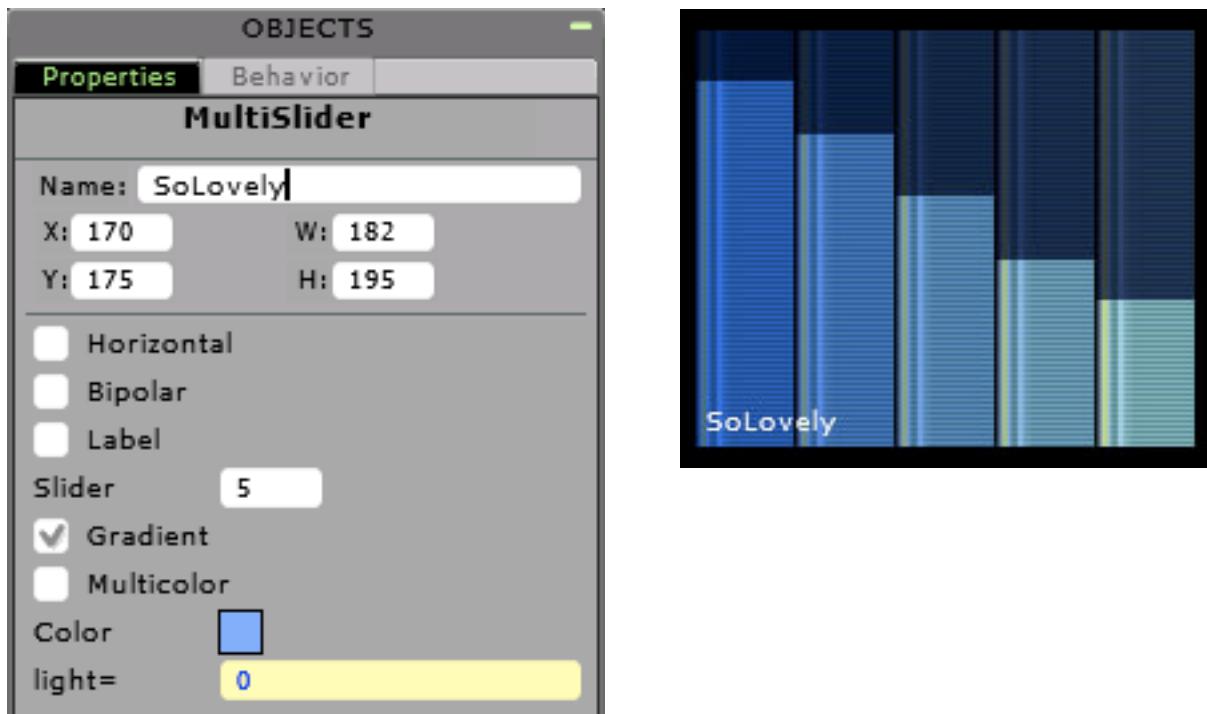
Change the **Attraction** value to 1.0 and the **Friction** to 0.2.



Try moving the ball now. It bounces off the "walls" and eventually slows down. The Mass-Spring mode is similar to Interpolation, except that the ball has friction (or a lack of it), can bounce off walls, and in certain cases, the ball may oscillate before coming to a complete rest. You can also control the speed of the ball via the **Speed** parameter.

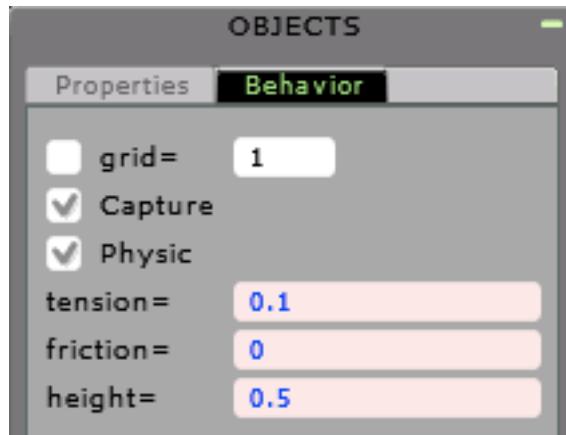
Physics parameters are great for producing complex time-varying values with little effort. We will see later on that we can obtain even more enjoyable effects by controlling the Physics properties with other objects.

As another example we create a MultiSlider Object with 5 sliders and name it *So Lovely*:



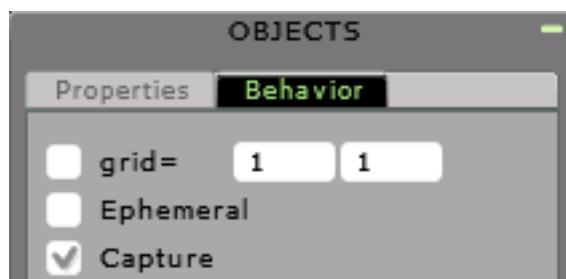
We then drag it to a free spot on the Interface, and make it bigger so we can interact easily with the individual sliders.

By default the MultiSlider Object doesn't display any physical behavior. The sliders simply follow your fingers and stay where you leave them.



Now change to the **Behavior** panel and activate the **Physic** checkbox. Also, change the tension to 0.1 and the friction to 0. Now it's a completely different story. If you move the sliders they will continue to wiggle forever. They behave as if they are connected with springs and oscillate around a centre value defined by the **height** parameter.

Again, we are only scratching the surface here. Please have a look at the Object Reference Chapter for details about all Object Properties.



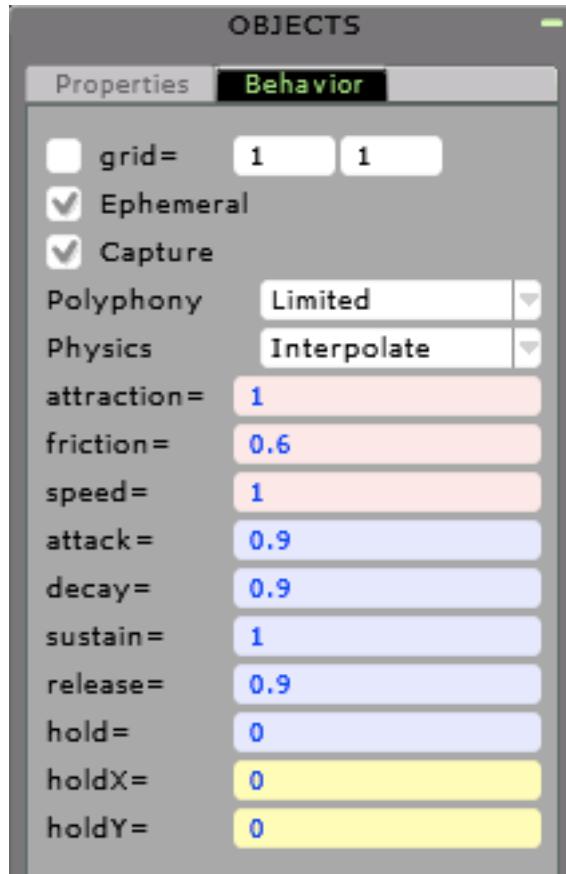
Let's come back to our MultiBall Object. The **Capture** parameter on the Behavior tab has a great influence on the possible finger actions. When you are inside of the Object's boundaries, the Capture mode doesn't make any difference. But if you move the ball and leave the borders of the Object, the control of the ball will stop when Capture is off. If Capture is on, the ball will continue being tied to your actions. No matter if you move your finger across other Objects and even if you switch to a different Interface, the ball will still cling to your finger and the finger has no effect on other Objects. Just experiment a bit with the two modes and you will quickly get the hang of it.

The MultiBall Object is a complex critter. It also sports an **ADSR+H** envelope for its brightness parameter or z Variable. You might be familiar with envelopes from using synthesizers. They represent an easy way to produce defined parameter changes over time.

The MultiBall ADSR acts just like its cousins from the synthesis world. When you touch the Lemur the brightness of the ball will be faded in according to the Attack value, decay down to the sustain level and fade out with the release time when you lift your finger from the surface.

Objects supporting ADSR envelopes (MultiBall and Pads for the time being) also have a **hold** parameter. Its effect is similar to a sustain pedal, freezing the object's state as long as its value is 1.

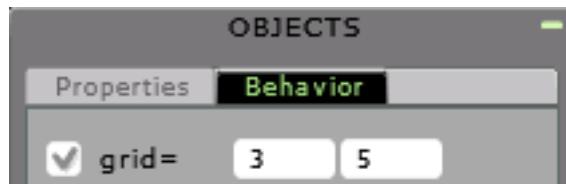
When set to 0, it has no effect. Interesting effects can be achieved when setting the hold parameter to an expression depending on other objects' state. We'll cover such methods later on.



The above picture is an example of brightness ADSR. Please note that the Ephemeral mode has to be active for the envelope to work. In **Ephemeral** mode the balls appear when you touch the surface and they vanish when you lift your finger.

Sometimes you don't want the full resolution of an Object because the target parameter in your software is quantized to only a few values. You can quantize the values sent via OSC by using variables and expressions, but the steps introduced by the quantization won't be reflected in the behaviour of the Object.

Enter the **Grid** parameter. This parameter is available for several objects, and "quantizes" the movement of your Object into multiple **steps**. The number of steps can be chosen in the adjacent fields. Test it with the *SoFunny MultiBall* object. Open its Properties and check the grid flag after setting its x-value to 3 and the y-value to 5.



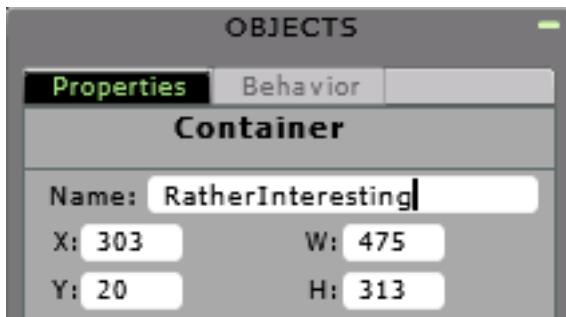
If you now have a look at the Object you see that the MultiBall area displays a grid and the Ball can only move to distinct points on each axis.



The values produced by the *SoFunny.x* and *SoFunny.y* variables are of course also quantized to produce only three steps on the x-axis (0.000, 0.5000, 1.000) and five steps on the y-axis (0, 0.25, 0.5, 0.75, 1).

## 6.9 Using Containers

The next thing we want to do is put some of the Objects into a Container to separate them visually from our Faders. To do this, drag a square around the MultiBall and the Switch Object to select them. Now right-click (ctrl-click on Mac) on the selection and choose **Cut** from the menu. The Objects disappear. They are not gone, though, but just got copied into the clipboard of the JazzEditor.

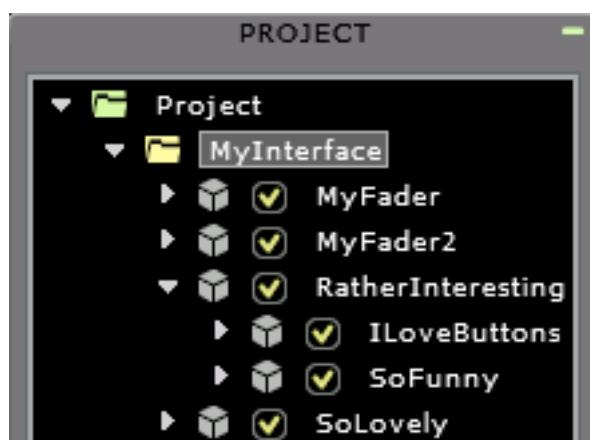


Click the **Create Object** button and pick a Container from the displayed list, or drag and drop it from the Palette. Name it and position the Container to the right of the Faders, then resize it to encompass the complete right part of the Interface.

Right-click (Command-click for Macs) the Container and choose **Paste** from the menu. This puts the two Objects we have cut out into the Container.

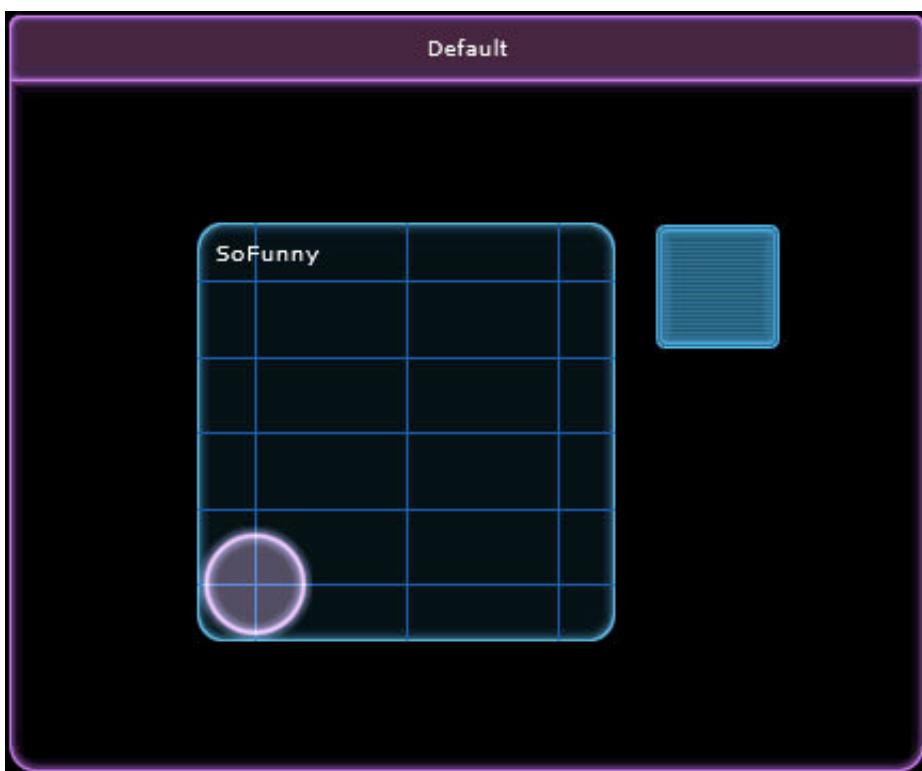
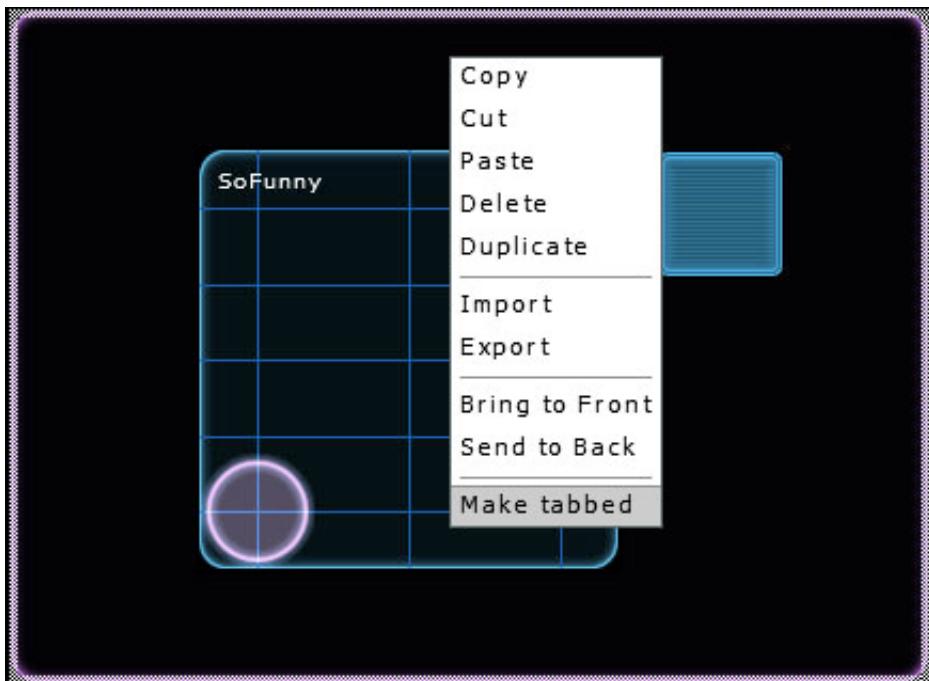


Now it's time to have a look at how all this is mirrored in the Project panel. The Container can be opened in the Project panel by clicking on the **disclosure triangle** in front of the Container. You see that the two Objects in the Container are grouped on a lower level in the Project hierarchy.

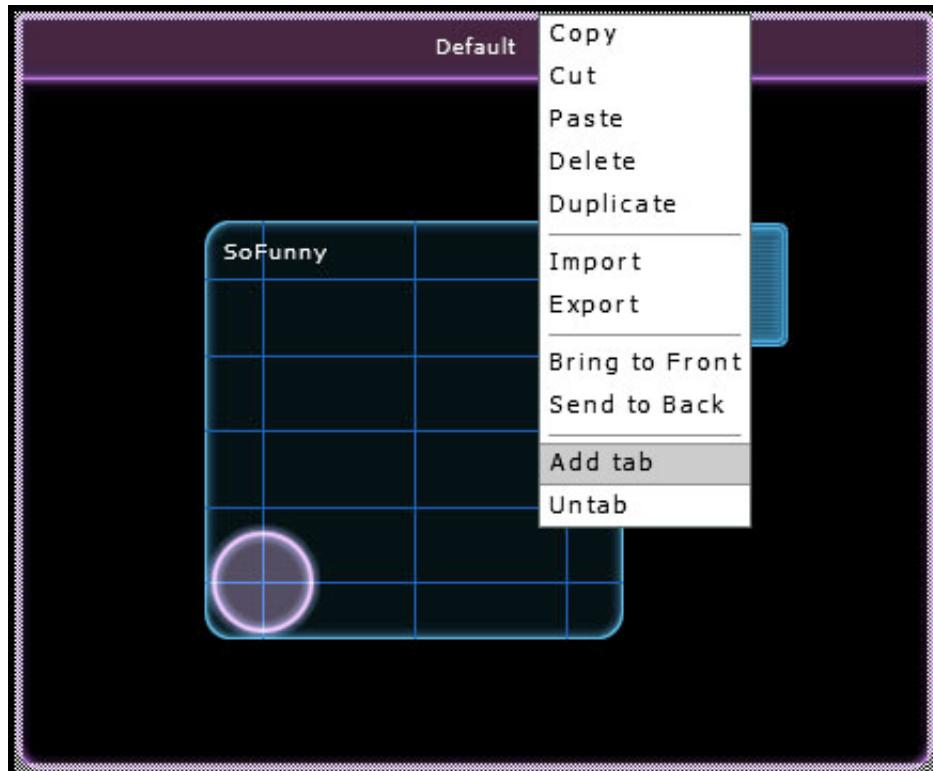


Note that the use of Containers also has implications for the Variable addresses of Objects living inside of the Container. Indeed, within the Container, the local names of both objects are still the same. From the outside however, there are now known as *RatherInteresting.ILoveButtons* and *RatherInteresting.SoFunny* respectively. Do you feel a little bit lost? Well, that's normal at this point, but don't worry, we will come back to this later.

A Container Object can also be made tabbed and additional tabs inserted in the Object. To make your Container tabbed, simply by ctrl-click (Mac)/ right-click (PC) onto the standard Container object and choose the option **Make tabbed**.



Additional tabs can then be inserted with the option *Add tab*:

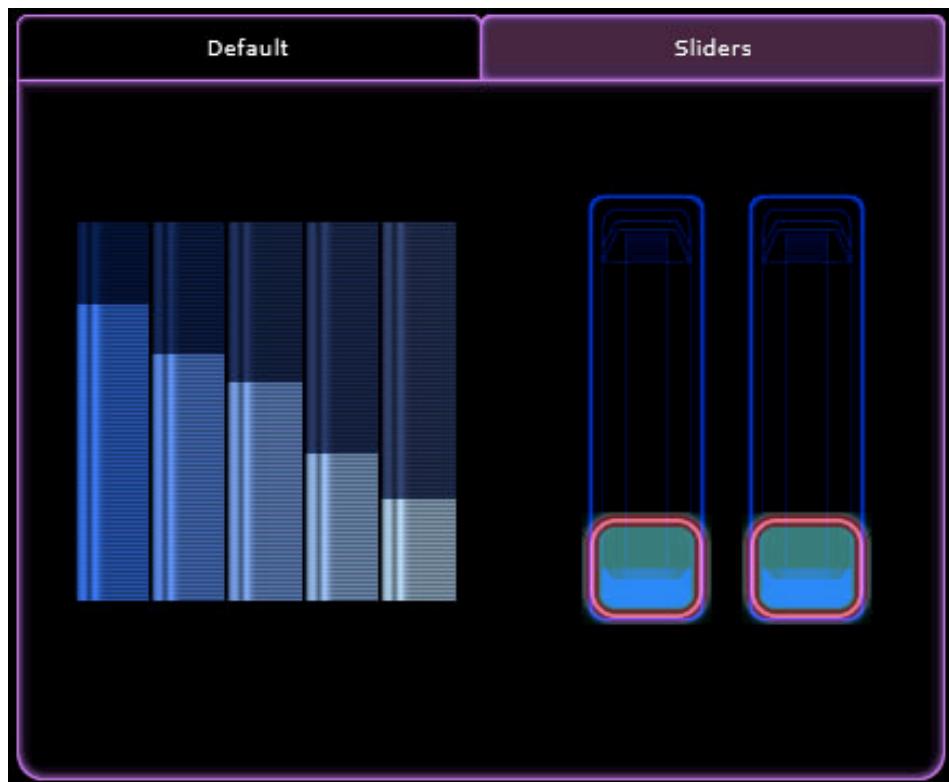


A small window then appears to name the new tab. Call it *Sliders* for instance :

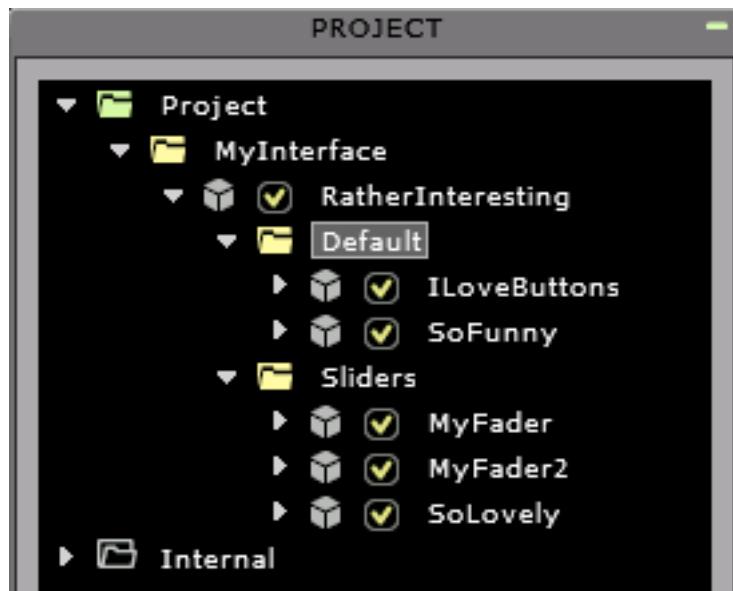




Now select the remaining Objects outside the Container, cut them and paste them inside the new tab:

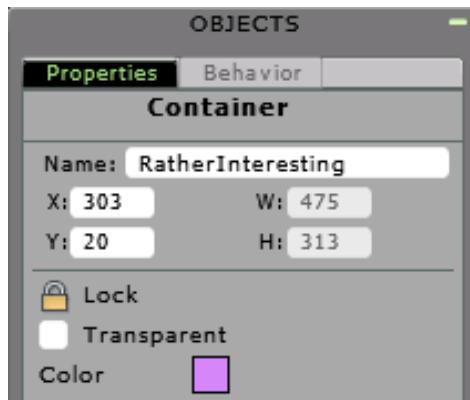


In the Project panel, each tab is represented as a folder inside the Container Object. To switch from one tab to the next in the JazzEditor, click on the desired tab in the Project panel, or enter into Run mode by holding the “e” key on your keyboard and clicking on the corresponding **TabBar**.

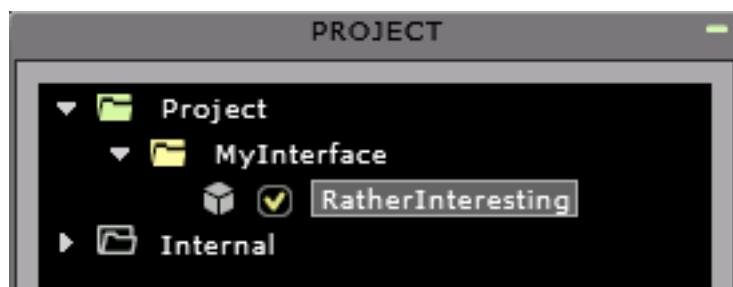


If needed, the tabs' order can be modified by clicking within a Container and choosing the **Bring up** or **Bring down** actions.

In large Projects the Project panel can quickly become crowded with Objects, leading to a lack of overview. In that case you may want to use the Container Object's **Lock** feature. Just try it with our *RatherInteresting* Container. Check the Lock flag in the Properties and you will see the MultiBall and Switch Objects vanish from the list.



A locked Container doesn't show its contents in the Project panel. This of course also means you can't edit the Objects inside of it. Simply unlock it again, if you need access to the contents.



## 6.10 Import and Export of Modules

Now let's suppose you would like to reuse later some portions of the interface in a different project. For that purpose, the JazzEditor's Toolbar provides the handy **Import Module** and **Export Selection** commands for the integration and creation of Modules. Any imported Module is incorporated into the currently opened Interface and you can select any group of Objects for export as a Module to the file system (the file extension for Modules is **.jzlib**).



Please note that you can also import and export Interface elements via the **contextual menu** (right-click/control-click on the Editing Area). Alternatively, you may drag your selection directly to the Library panel while holding the **alt** key, provided you've defined a path for your Library files in the JazzEditor's settings page.

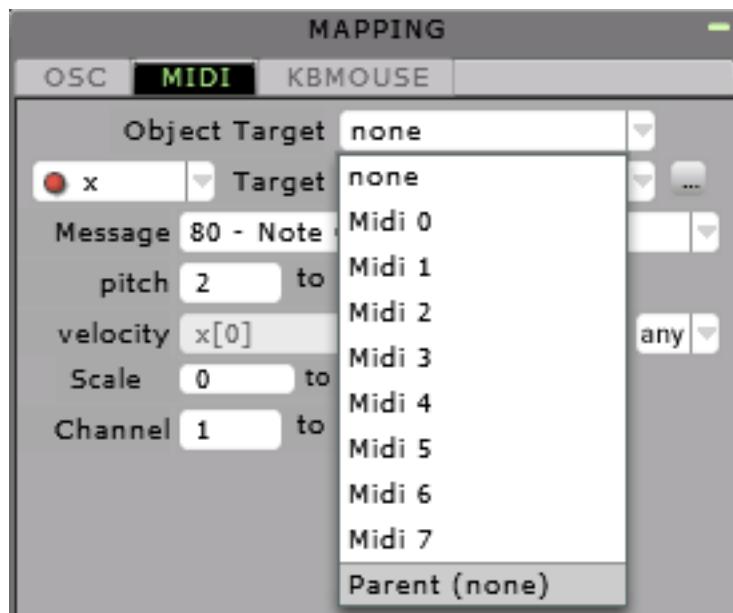
Make sure that you do not destroy any dependencies in terms of Variables and expressions when you export parts of your Projects. It is good practice to encapsulate the exported Module in a Container and have the variables and functions that are important for the Module's functionality created locally to that Container. If you define them globally, they will not get exported and the functionality of the Module will be broken.

# Chapter 7 - Mapping

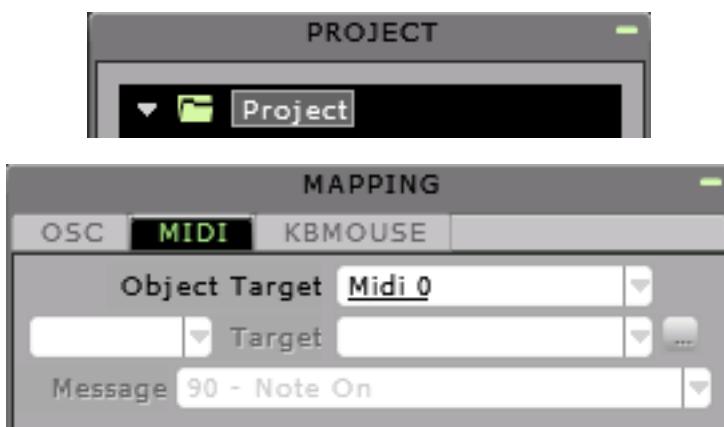
## 7.1 Setting up MIDI messages

The simplest way to assign MIDI messages to your Objects' Variables is to use the Mapping panel's MIDI tab.

The top of the MIDI panel features the **Object Target** menu, which lets you select a default MIDI Target (communication path to an application) for all built-in or User defined Variables local to the Object. You can choose *none*, any of the 8 available MIDI Targets, or **Parent**, a Parent being a higher hierarchy level in the Project panel's tree. An Object's Parent can be a Container or the Project itself. A Variable's Parent can be an Object, a Container or the Project itself...

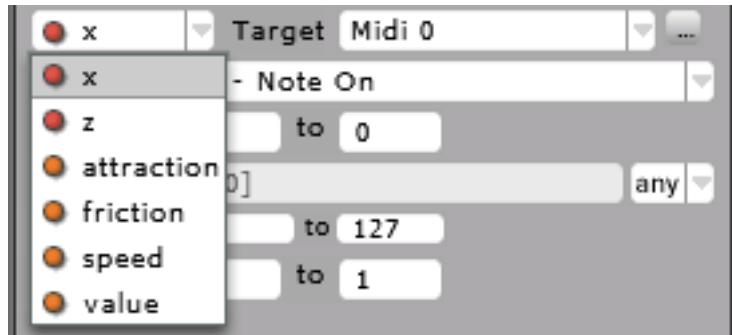


If an Object Target has been set up at a higher hierarchy level, this will be stated in the pull-down menu inside the brackets next to *Parent*. Let's check it out and set up a Target at the Project level. To do that we select the Project folder in the Project panel, and set its Object Target in the Mapping panel to, say, Midi 0:



Now, for any object created prior to this, the Parent will appear in the Object's Target menu. As for any new Object, it will automatically be set to the Parent's Target.

Below the Object Target's menu, you find the **Variable menu**, for choosing a local Variable to pair with a MIDI message, and the **Variable's Target menu**, for selecting a Target distinct from the Object Target if needed.

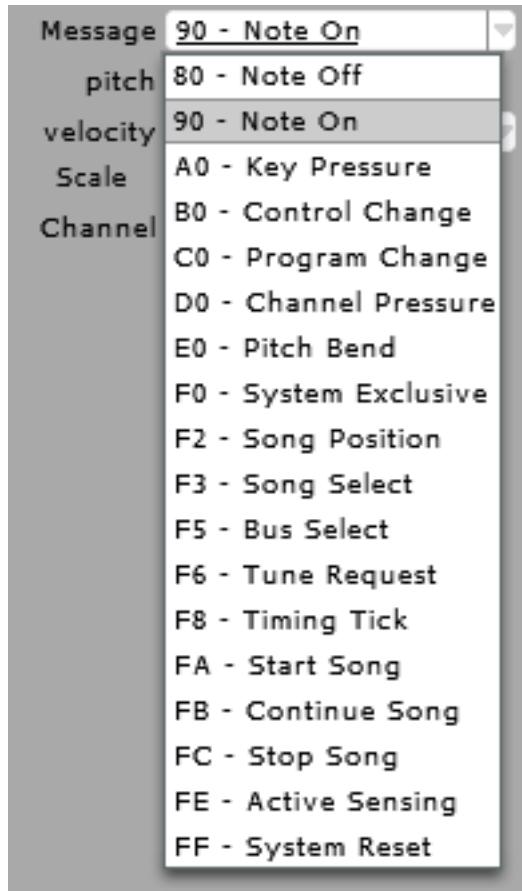


Clicking on the **Quick Map** button (the tiny icon next to the Variable's Target menu) automatically sets the Variable's Target and MIDI message selection to those chosen in the JazzEditor settings window's Lemur tab for the **Automap** feature. This can be thought of as a handy “semi-automatic” mapping function.

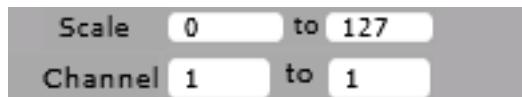


The next element is the **MIDI Message menu**, whose purpose is, quite naturally, to let you select the MIDI message you want to pair with your Variable. Messages range from Note messages over Controllers to System Realtime messages, like Song Start/Stop or Active Sensing (the different messages type are preceded by their MIDI status byte in hexadecimal form). In most cases though, you will either choose a Control Change or a Note On MIDI message.

Depending on the chosen message type, the MIDI tab displays different parameters to tweak, such as “pitch” for Note On messages, or “controller” for Control Change messages. Please refer to the MIDI Message reference on Chapter 15 for details.



The **Scale** parameter, typically set from 0 to 127, ensures that the 0-1 values generated by the Object's Variables are scaled to the MIDI values, which are typically integers (whole numbers) between 0 and 127



It is important to understand that the communication with MIDI Targets set up in the MIDI tab is **bidirectional**. In other words, a Variable to which a MIDI message has been assigned in the MIDI tab can transmit this message but will also react to the same incoming message. This explains why you also find Variables in the Variables menu that do not output values.

By the way you might already have noticed the different colours for the Variables in the menu:

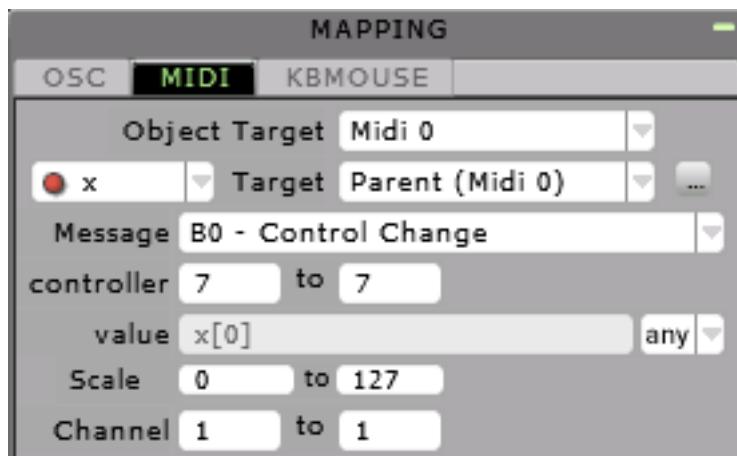
- **Red** stands for the Object's main parameters (x for a Fader; x, y, z for a MultiBall, etc.)
- **Orange** depicts a properties/behaviour parameter (like friction, attraction, value, etc.)
- **Green** entries are Expressions created by the user.

The use of the **Trigger Mode menu**, common to three Mapping panel's tabs, is detailed later on in this chapter. It allows for choosing when and how often the messages are actually transmitted (triggered).

## 7.2 Simple MIDI mapping examples

Let's have a look at a simple example:

- Select a Fader in your project. To assign a MIDI message to this fader, click its MIDI tab within the Mapping panel.



- Change the settings of the dialog to those shown above (Message is Control Change, controller number is 7, and value is x). The Scale fields should be set from 0 to 127. Set the MIDI channel according to the channel of the desired Target, which is chosen via the Target Menu to the upper left. These settings configure the Fader to transmit MIDI controller 7 (volume) with the fader's 0-1 floating-point values scaled to 0-127.
- If you wish to, you can click on the **MIDI Map** button located on the JazzEditor's Header to verify the MIDI assignment:



The MIDI Mapping window then appears, displaying the assignment we just configured:

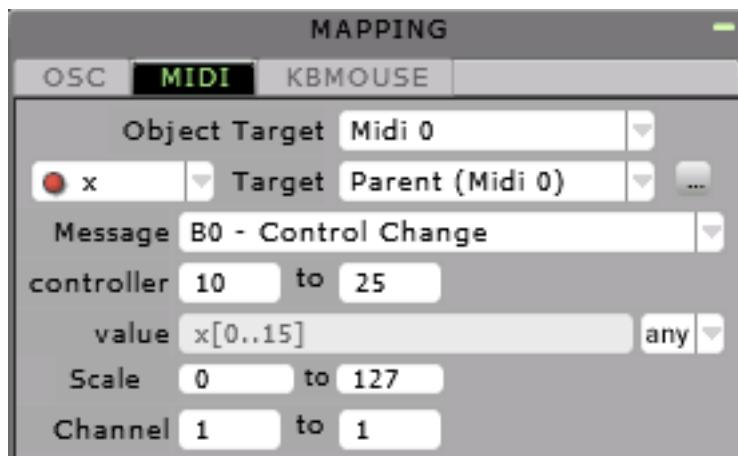
Name	Msg	Ch	Message Name	Param 1	Param 2	Target
Fader.x	B0	1	Control Change	7	x	Midi 0

In the next example, we'll use a MultiSlider object and automatically assign different control change messages for each slider.

- Create a MultiSlider object. Using the Properties tab, set the number of sliders to 16.



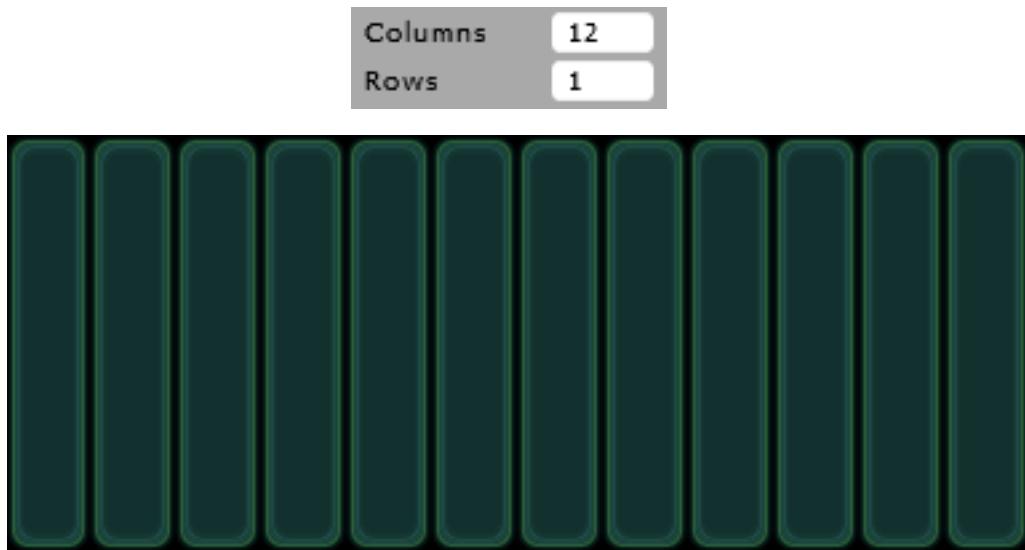
- Now click on the MultiSlider's MIDI panel



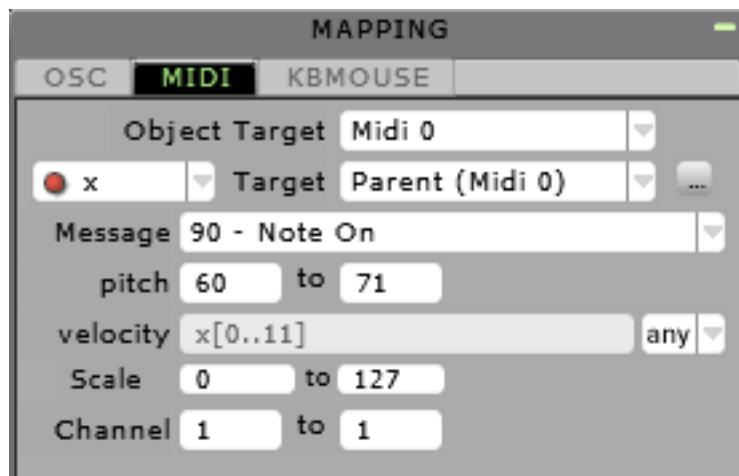
- Choose **B0 Control Change** and **MIDI 0** as Target for the x Variable.
- Type controller number **10** into the first controller field. The second controller field is automatically set to **25** to extrapolate for the 16 sliders. Indeed the JazzEditor automatically assigns each slider to successive MIDI controller values. The same thing happens for other multi-value Lemur variables (such as MultiBall's parameters).

The following example uses the same automatic mapping feature to create a chromatic one-octave MIDI “pad” on your Lemur that will send MIDI note messages.

- Create a Pads object. Give it 12 columns and one row.



- Select the Object and go to the Mapping panel’s MIDI tab. Choose the x Variable and Note On as the MIDI message type.
- Set the first pitch field to 60 and the second will automatically be expanded to 71 (reflecting your 12 Pads). Your MIDI assignment should resemble the one shown below.



- Click the MIDI Map icon to look at the MIDI Mapping window. You see the 12 Pads nicely laid out working as a MIDI keyboard.

Midi Mapping							
Name	▼	Msg	Ch	Message Name	Param 1	Param 2	Target
Pads.x		90	1	Note On	60	x	Midi 0
Pads.x		90	1	Note On	61	x	Midi 0
Pads.x		90	1	Note On	62	x	Midi 0
Pads.x		90	1	Note On	63	x	Midi 0
Pads.x		90	1	Note On	64	x	Midi 0
Pads.x		90	1	Note On	65	x	Midi 0
Pads.x		90	1	Note On	66	x	Midi 0
Pads.x		90	1	Note On	67	x	Midi 0
Pads.x		90	1	Note On	68	x	Midi 0
Pads.x		90	1	Note On	69	x	Midi 0
Pads.x		90	1	Note On	70	x	Midi 0
Pads.x		90	1	Note On	71	x	Midi 0

Refresh

Close

You can use more than 12 pads to create a bigger keyboard if you wish.

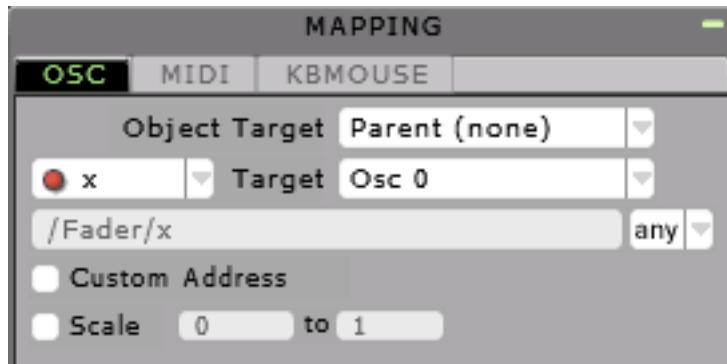
The Midi Mapping Window consists of seven columns:

- The **Name** column shows the name of the Variables used.
- The **Msg** (Message) column displays the numbers of the different MIDI messages used for the individual mappings.
- In the **Ch** (Channel) column you find the associated MIDI channel for the individual mappings.
- The **Message Name** column writes out the full name of the different MIDI messages.
- The two **Parameter** columns show the MIDI parameters associated with the chosen message type, i.e. the CC number and the CC value for a MIDI controller message.
- The **Target** column shows which of the MIDI Targets that are defined on the Settings window have been chosen for the respective mapping.

The **Refresh** button updates the list of MIDI mappings and you can close the window by hitting the **Close** button or the escape key on your computer keyboard.

## 7.3 OSC

The **Open Sound Control** protocol specifies the transmission of messages between two devices. Rather than attaching specific meanings to these messages - as it is done with MIDI - Open Sound Control allows you to define your own system of messages. With the Lemur, the names of objects you create and their “path” in the Project hierarchy constitute their default “address” for OSC messages.



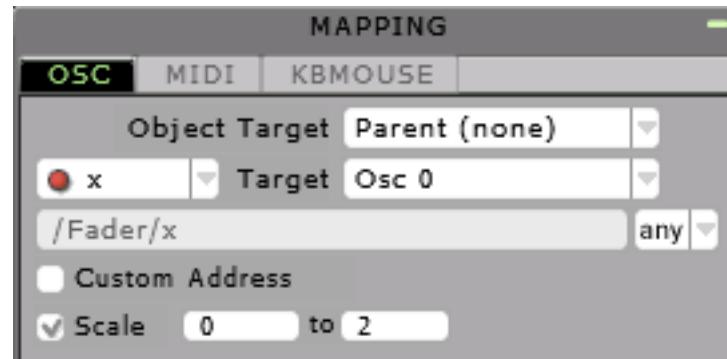
As in the MIDI tab, the top of the OSC tab features the **Object Target menu** which lets you select a default OSC Target (communication path to an application) for all built-in or User defined Variables local to the Object.

On the next line, you see the **Variable menu** and to its right the **Variable’s Target menu**. Let’s say we want to transmit the Fader’s x Variable via OSC. If you pull down the Variable menu, you see that it lists all available Variables of the Fader: Choose the x. From the OSC Targets menu, choose the Target that you have set up in the general OSC Settings.

If you now move your Fader it will send the value of x to the OSC address /Fader/x. Use whatever OSC software you have to check it.

If you want to change the OSC address use the **Custom Address** flag and type in whatever address you need. This may be useful in case of naming conflicts between Lemur Projects and projects running on the Target side.

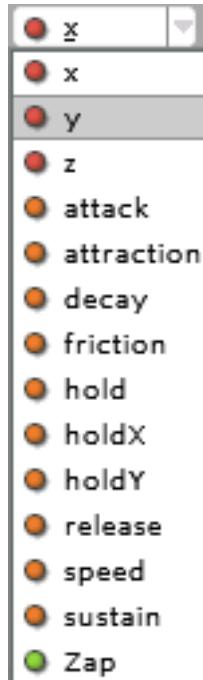
Default parameters of objects range between 0 and 1 (such as the x variable of a Fader). However, you can choose to scale their OSC output to your own range. Enable the **Scale** checkbox and enter other integer boundaries for the output.



The communication with OSC Targets is **bidirectional**. You define the pairing for both incoming and outgoing OSC data with the various parameters of your Objects on the OSC tab. That’s why you also find Variables on the menu that don’t output values: they can be remote controlled from the displayed OSC address.

Main object's parameters react to incoming OSC according to their OSC range. Sending /Fader/x 1.0 to the Lemur will bring the Fader's cap to the top if the default range is used. If you've customized the range to 0..2 as explained above, the Fader's cap will instead go midway, since 1.0 is the centre of the 0..2 range.

Note that OSC scaling is only enabled to object's main parameters, and not to user-created expressions, since those have undefined range by essence.



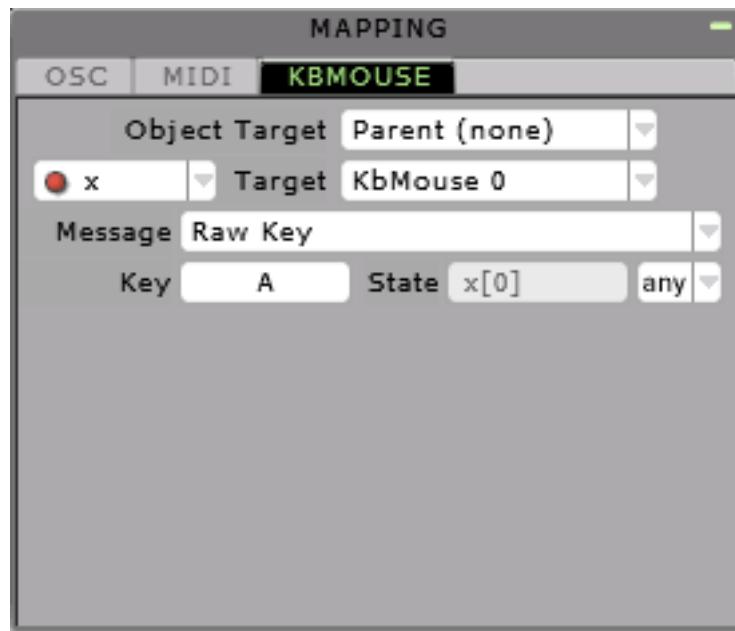
You might already have noticed the different colours for the Variables in the menu. By the way, the above menu is from a MultiBall Object.

- **Red** stands for the Object's built-in Variables (x for a Knob; x, y, z for a MultiBall, etc.)
- **Orange** depicts a properties/behaviour parameter (like friction, attraction, value, etc.)
- **Green** entries represent User-defined Variables.

The use of the Trigger Mode menu is detailed later on in this Chapter. It allows for choosing when and how often the messages are actually transmitted (triggered).

## 7.4 Keyboard and Mouse

The **KBMOUSE** mapping tab works along the same lines as the MIDI and OSC tabs, but its purpose is to let you assign keyboard commands, be they single key strokes or combinations, and mouse movements or clicks to your Objects' Variables. A basic use of this tab would be to assign a keyboard shortcut to a Pad Object for instance, in order to control an application that does not feature any MIDI or OSC compatibility. Another use would be to take over your mouse with a Gesture object, and integrate a Trackpad within your interface.



As in the MIDI and OSC tabs, the top of the KBMOUSE tab features the **Object Target menu** which lets you select a default KBMOUSE Target, that is, a computer with a JazzDaemon running, for all built-in or User defined Variables local to the Object.

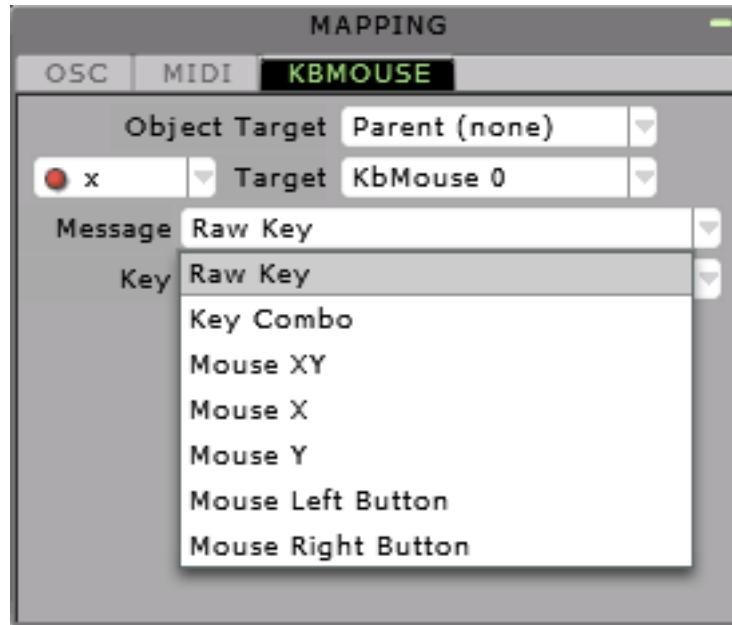
Below the Object Target's menu, you find the **Variable menu**, for choosing a local Variable to pair with a MIDI message, and the **Variable's Target menu**, for selecting a Target distinct from the Object Target if needed.

The next line features the **Message menu**, for selecting the type of command you wish to pair with your Variable. The choice is as follows:

- **Raw Key:** for a single key stroke, defined by the key typed in the **Key** field and triggered by a change in the Variable's **State** according to the chosen **Trigger mode**.
- **Key Combo:** for a combination of simultaneous key strokes, defined by the key combo typed in the **Combo** field, and triggered according to the chosen **Trigger mode**.
- **Mouse XY:** for a 2D mouse movement paired to a vector Variable {x, y} and triggered according to the chosen **Trigger mode**.
- **Mouse X:** for a horizontal mouse movement paired to the Variable's value and triggered according to the chosen **Trigger mode**.
- **Mouse Y:** for a vertical mouse movement paired to the Variable's value and triggered according to the chosen **Trigger mode**.

- **Mouse Left Button:** for a left click triggered by a change in the Variable's **State** according to the chosen **Trigger mode**.
- **Mouse Right Button:** for a right click triggered by a change in the Variable's **State** according to the chosen **Trigger mode**.

The use of the Trigger Mode menu is detailed in the next section.



## 7.5 Trigger Modes

The purpose of the Trigger Mode menus is to define when a message should be transmitted with respect to a change in a Variable's state.



**any:** The message is sent whenever the parameter changes.

**up:** The message is sent each time the parameter rises from 0 into the positive value range.

**down:** The message is sent each time the parameter reaches 0 from the positive value range.

**up and down:** The message is sent each time the parameter reaches 0 OR rises from 0.

**+**: The message is sent each time the parameter increases above its previous value.

**-**: The message is sent each time the parameter decreases below its previous value.

For Custom MIDI Messages (see Chapter 10 for an explanation) there is an additional item on the Trigger Mode Menu:



The first entry ("None") means that a change in the attached value or expression will not trigger a transmission of the MIDI message. It might, however still get triggered by values or expressions associated with other parameters of the Custom MIDI message that don't have their Trigger Mode set to None.

# Chapter 8 - Targets Setup

## 8.1 JazzDaemon Targets

The JazzDaemon handles the MIDI data flow between the Lemur and the MIDI ports installed on the computer, as well as any mouse movements or ASCII commands through the KbMouse port.

The JazzDaemon automatically scans all available MIDI ports on the computer, be they physical or virtual, and make them available to the Lemur(s) connected to the network. On Mac, the JazzDaemon also creates its own set of virtual ports, the Daemon inputs and outputs, which are accessible in your applications' MIDI settings. On PC, third party virtual ports such as those offered by MIDI Yoke or Maple are needed in order to link to the JazzDaemon and use MIDI with your Lemur.

It doesn't end there. You can also have several computers on the same network all running JazzDaemons and all available MIDI ports and KbMouse ports of all computers will be accessible from JazzDaemon Targets setup page of your Lemur. This means maximum flexibility in terms of MIDI topology. Regardless of the complexity of your MIDI network, you can reach every single device via your Lemur, as long as it is connected to a computer on the same network as the Lemur.

JazzDaemon Targets can either be set up on the Lemur itself or via the JazzDaemon running on a computer.

There are some thoughts to be invested about being able to make MIDI/ KbMouse connections on the Lemur and making them from the JazzDaemon itself. The two methods lead to different behaviour when starting the Lemur or the Daemon, respectively:

- Connections created from the Lemur can be saved in its internal Flash from the **Save Settings** button. The Lemur will then automatically attempt to reconnect to the Daemons at boot. If the connections can't be re-established (due to a missing Daemon on the computer), the entries on the Lemur's JazzDaemon Targets settings are greyed out. As soon as the awaited Daemon appears on the network, the Lemur makes the corresponding connection. If you always connect your Lemur to the same computer(s), saving the JazzDaemon Targets connections on the Lemur is recommended, as this may be timesaving.
- Connections created from the JazzDaemon itself are saved in its preferences file. They will be automatically remade when the JazzDaemon is launched. If the connections can't be re-established (because the Lemur is connected), the entries on the Daemon's settings are greyed out. As soon as the awaited Lemur appears on the network, the Daemon makes the corresponding connection. If you need to have several JazzDaemon setups depending on which computer the Lemur is hooked to, it is recommended to create the connections from the JazzDaemons: this way, whenever you plug your Lemur to any of those computers, the relevant connections will be made.

On your Lemur:



The JazzDaemon Targets configuration screen on the Lemur displays the currently defined **MIDI Targets (two-way)** and **KbMouse Target (one-way)**. There can be up to eight of each kind. The MIDI communication is bi-directional, meaning you can choose a MIDI input port (incoming to the Lemur) and a MIDI output port (outgoing from the Lemur). Keyboard and Mouse communication is one-way only (outgoing from the Lemur to a computer). The ports available in the menu depend on the ports detected by the JazzDaemons currently running on the network.

Touch the Input or Output of a Target to be changed on the MIDI screen of the Lemur. You get a list of currently available Daemons on the network.

## 8.2 JazzDaemon Targets Setup Example

Let's go through the steps to get a MIDI connection up and running:

- On PC, start the JazzDaemon by going to the JazzMutant Start menu folder and clicking on the JazzDaemon icon. On Mac, click on the JazzDaemon icon in your Applications folder.



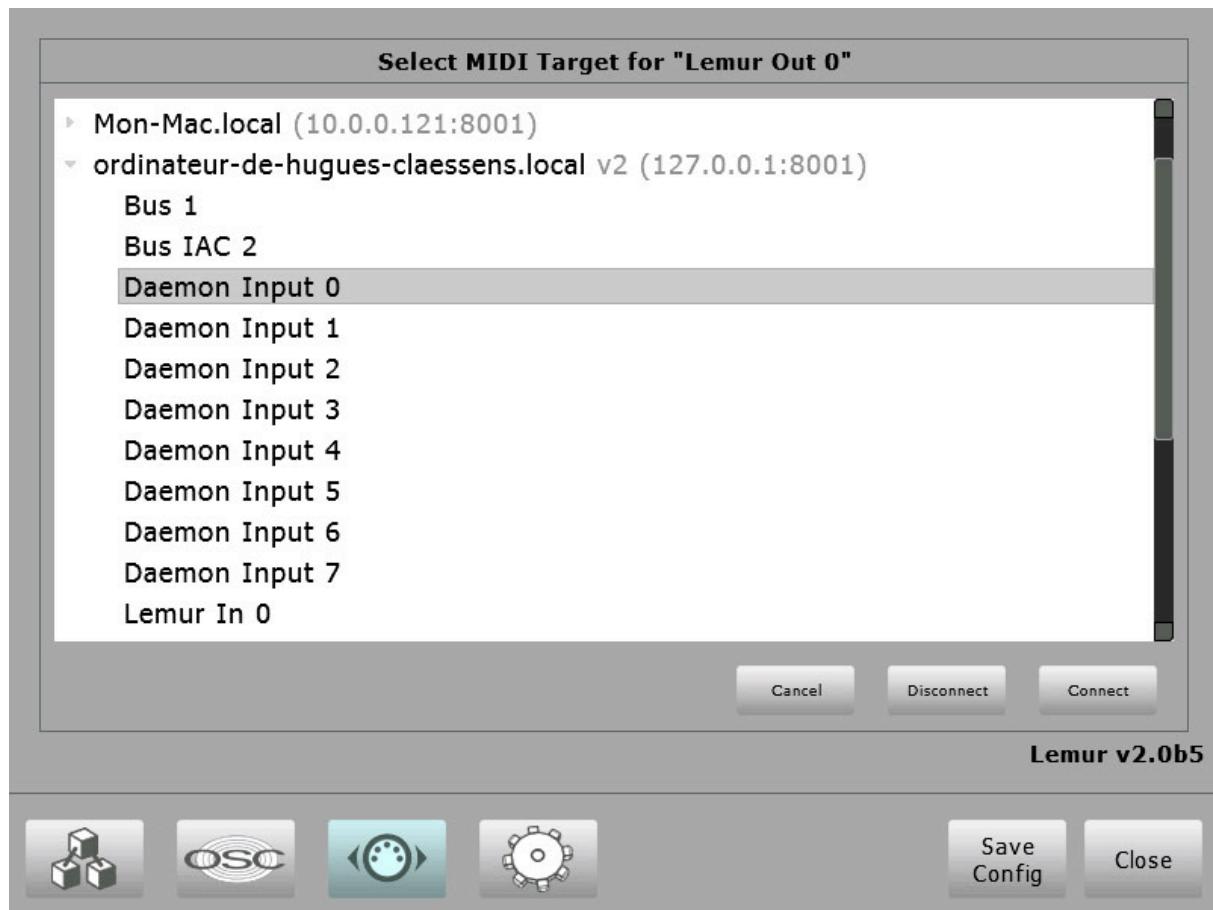
- Push the **Settings** button on your Lemur (top left hardware button) and touch the JazzDaemon Targets configuration button on the screen.

JazzDaemon Targets			
	Inputs	Outputs	KbMouse
0	none	none	none
1	none	none	none
2	none	none	none
3	none	none	none
4	none	none	none
5	none	none	none
6	none	none	none
7	none	none	none

Lemur v2.0b5

Close

- Now touch the **Outputs** button for the JazzDaemon Target 0 and see the list of Daemons detected by the Lemur.



- Touch the small triangle in front of the computer name to drop down the ports list. Choose one of the MIDI ports (see below for details) and touch the Connect button. You should now see the selected port appearing in the Targets list.
- You should now see the selected port appearing in the Targets list.

**JazzDaemon Targets**

	Inputs	Outputs	KbMouse
0	none	ordinateur-de-hugues-cla... : Daemon Input 0	none
1	none	none	none
2	none	none	none
3	none	none	none
4	none	none	none
5	none	none	none
6	none	none	none
7	none	none	none

Lemur v2.0b5

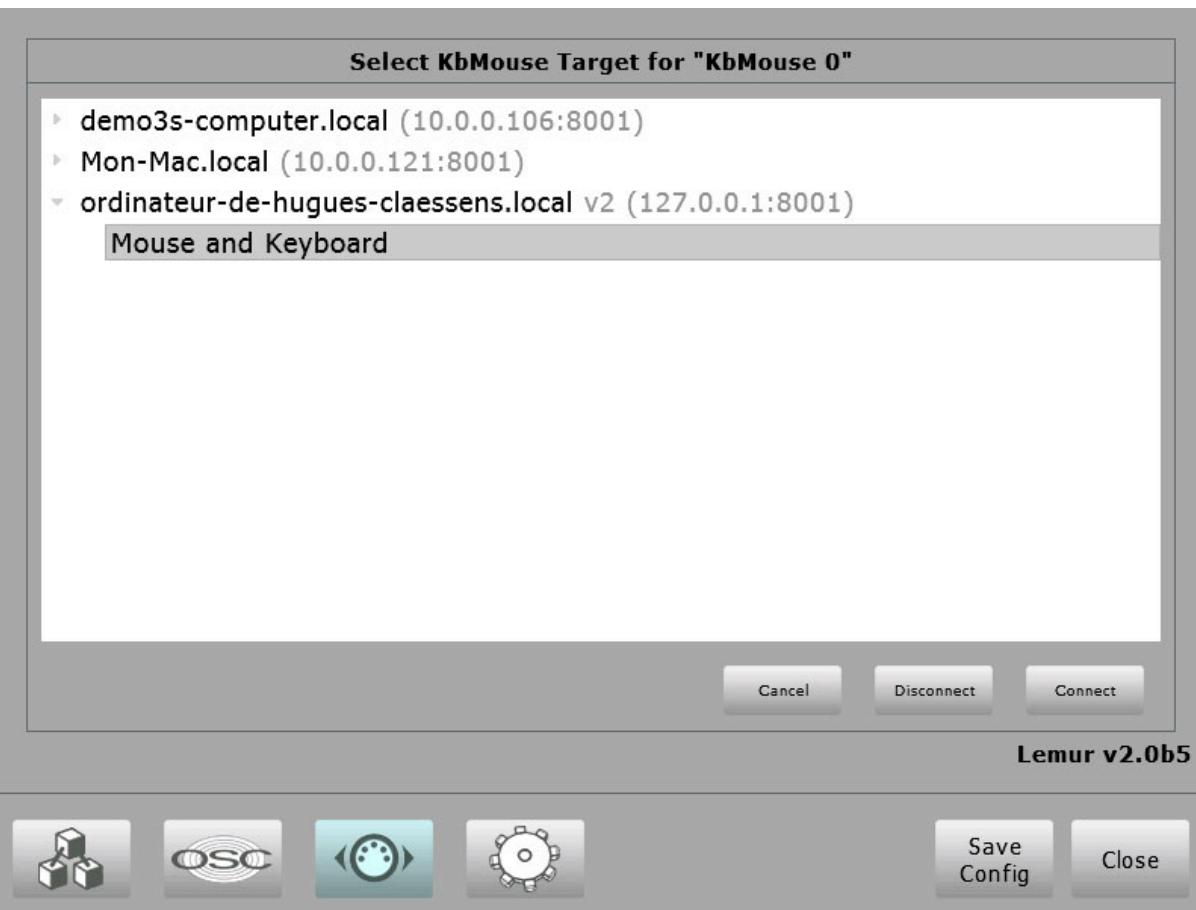
---








- Similarly, if some of your Objects on your Interface send Keyboard and Mouse commands, touch the **KbMouse** button for the JazzDaemon Target 0 and see the list of Daemons detected by the Lemur.
- Touch the small triangle in front of the name of the computer you want to control, select its **Mouse and Keyboard** and touch the **Connect** button.



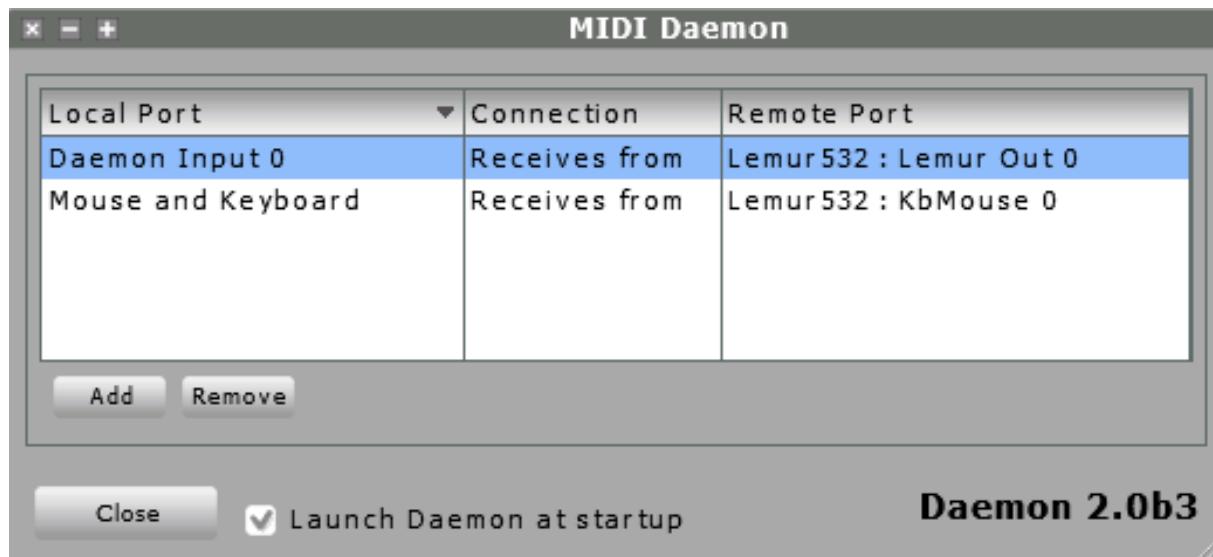
JazzDaemon Targets

	Inputs	Outputs	KbMouse
0	none	ordinateur-de-hugues-cla... : Daemon Input 0	ordinateur-de-hugues-cla... : Mouse and Keyboard
1	none	none	none
2	none	none	none
3	none	none	none
4	none	none	none
5	none	none	none
6	none	none	none
7	none	none	none

Save Config   Close

Lemur v2.0b5

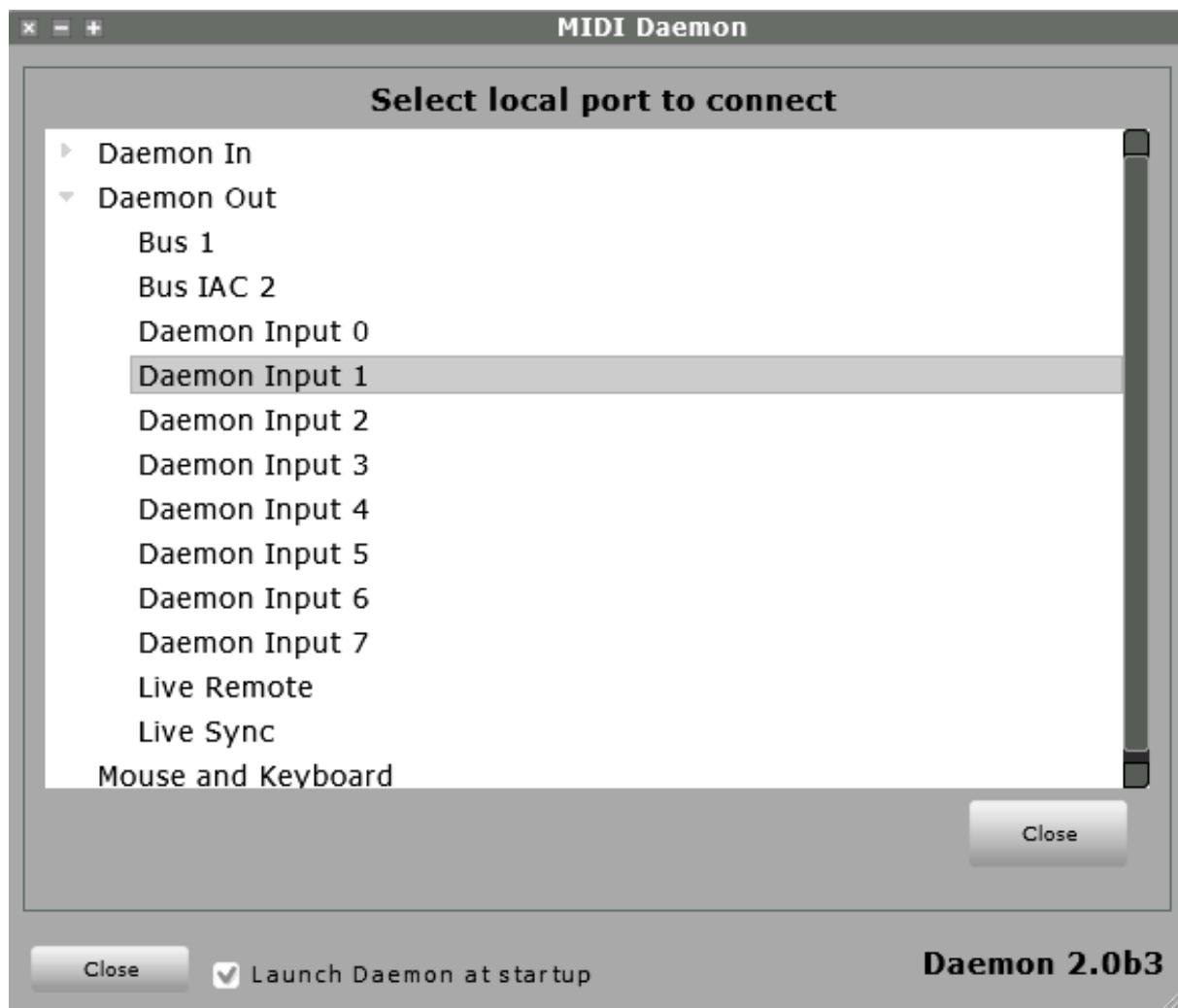
- When you're done, touch the **Save Config** button to save your configuration.
- Now let's see how the JazzDaemon register these connections. Click on the JazzDaemon's icon on your computer and choose **Setup JazzDaemon**.
- A window appears, listing the ports connections we've just configured.



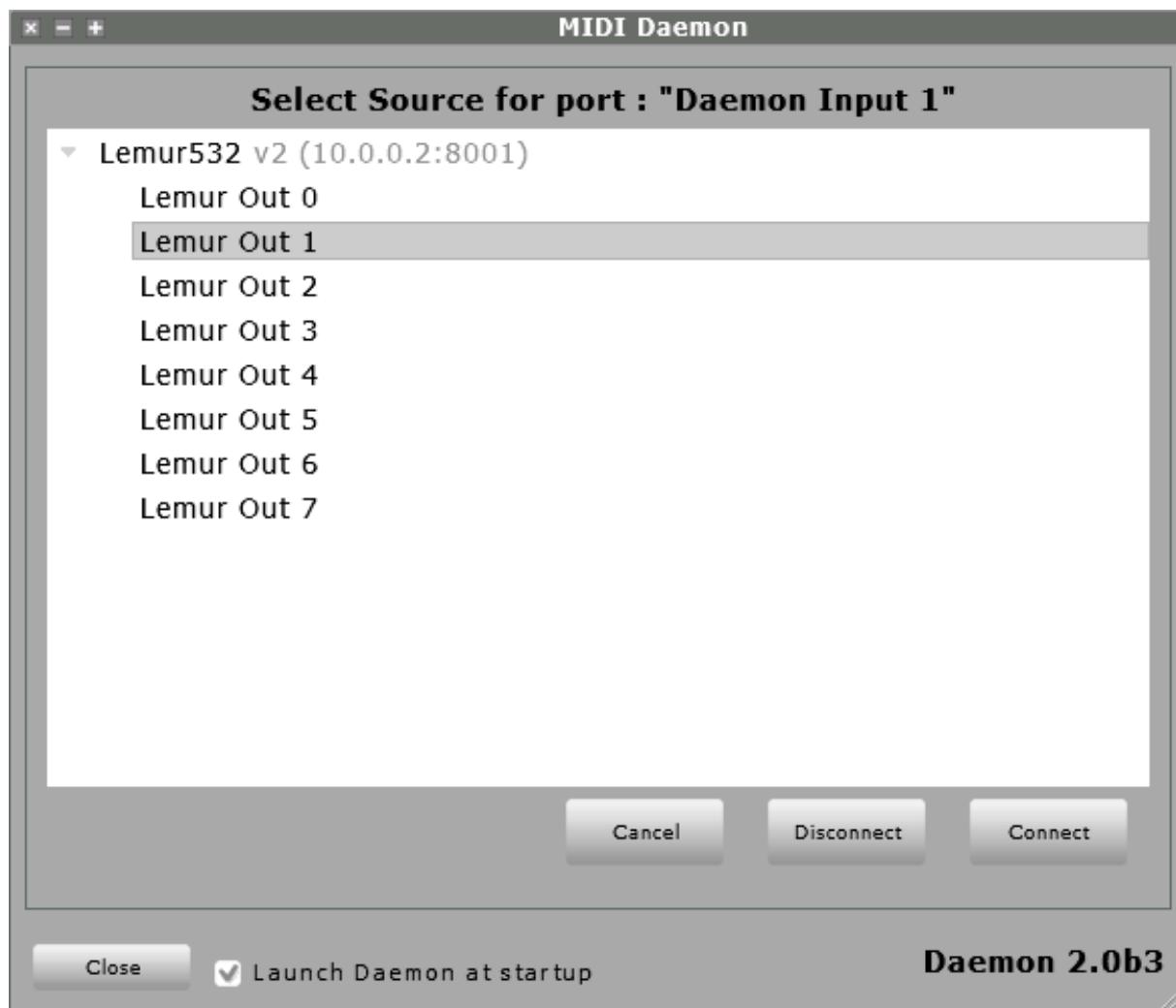
- You can now check if the MIDI connection works. Connect an application to the virtual MIDI port. Create a Fader and send some controller data to Target 0 or take a Pad Object and send some Note On messages.
- Turn your Lemur off.
- Turn it back on and have a look at the JazzDaemon Targets configuration page on your Lemur and at the Settings of the Daemon. Everything is still there and works as before.

Connections to MIDI or KbMouse ports can also be configured from the JazzDaemon:

- Click the **Add** button in the JazzDaemon window. You are now presented with all the MIDI ports and the Mouse and Keyboard port of the machine. You can choose any In or Out port to connect to a Lemur. (Note that Mac Users will see there the Daemon Input and Daemon Output virtual ports that are automatically created by the JazzDaemon at launch). Any port in the Daemon In section can be connected to the output of a Lemur. Any port in the Daemon Out section can be connected to the input of a Lemur.



- Double click on the local port you want to connect. Here, we want to receive all MIDI data coming from the Lemur's Target 1 on our virtual port named Daemon Output 1. You can now browse the network and look for a Lemur to connect to. Choose Lemur Out 1 as a source, and double click the item, or hit the **Connect** button.



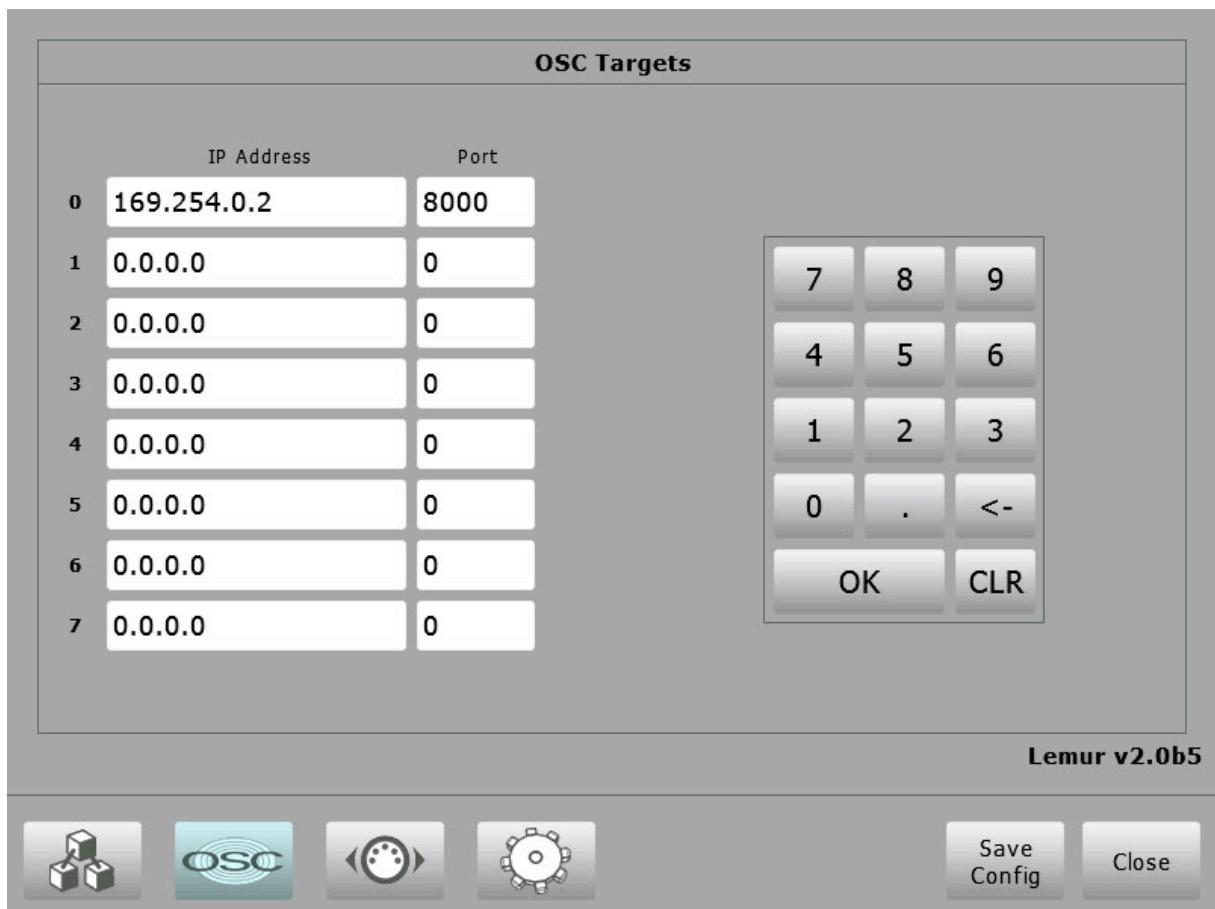
- Congratulations, you have successfully created a connection from the JazzDaemon! Now if you quit and re-launch the Daemon, or reboot your computer, it will automatically redo the connection from IAC 1 to Lemur Out 0. Use this technique if you often swap computers connected to your Lemur, and need a unique configuration for each of those.

## 8.3 OSC Targets

OSC Targets can be software applications on your local computer, somewhere on the network, or OSC-enabled hardware connected to the network. They are called Targets because they are what the Lemur targets its messages at. Always remember that OSC is bi-directional and that OSC Targets can also control your Lemur's Objects.

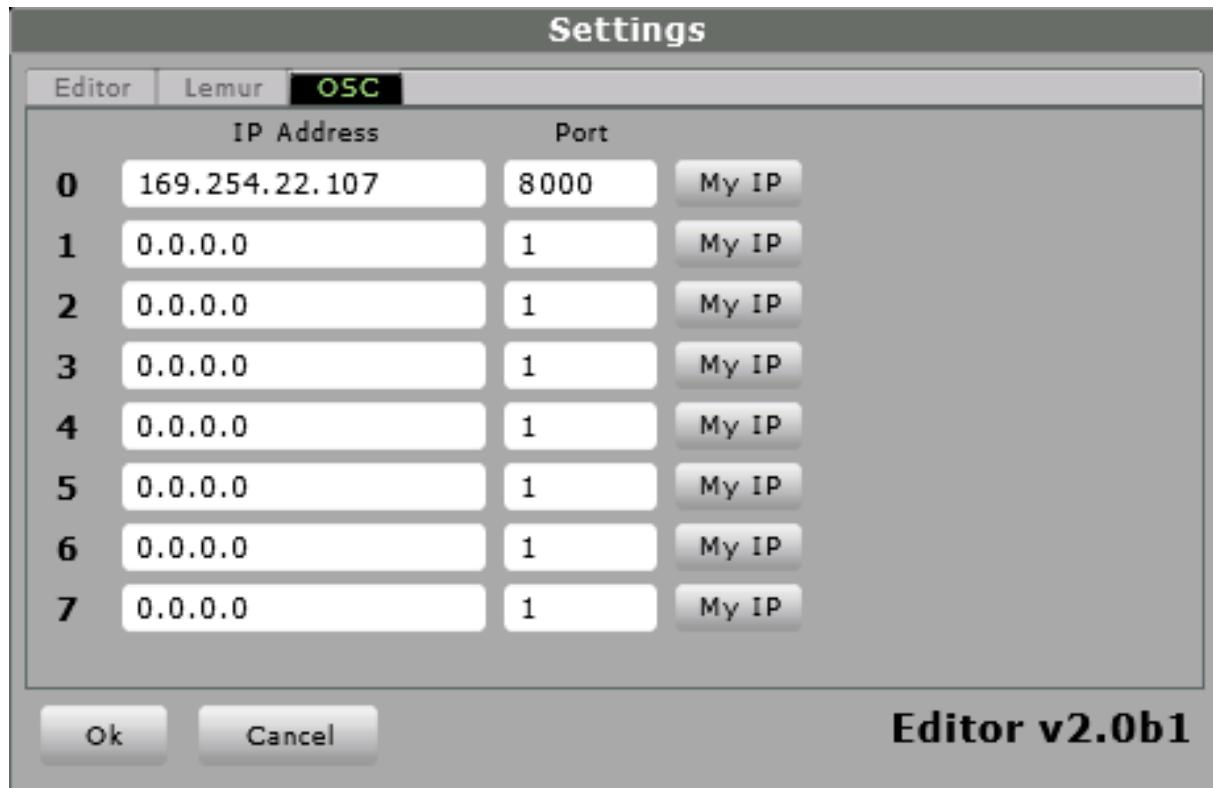
OSC Targets can be configured from your Lemur's **OSC Targets** configuration screen, which is the second page of your Lemur's Settings. OSC Targets are also mirrored in the JazzEditor's settings window if a Lemur is connected. You can use both screens to set up your OSC Targets.

On your Lemur, simply fill in the **IP Address** and **Port** number via the appearing number pad for each Target you want to communicate with.



Press the **Save Config** to save your OSC settings into the Lemur's flash memory. Whether you set up your OSC targets from the JazzEditor or from the Lemur, they will always be saved to memory and available at each reboot. The **Close** button closes the Settings window.

If you're configuring your OSC Targets from the JazzEditor's settings window, and if your Target is a piece of software running on the same computer as the JazzEditor, click on the **My IP** button to automatically fill in the IP of your computer.



The OSC Targets' **Port** setting is dependent on the corresponding setting in the targeted application. Ports are something like channel selectors that allow multiple applications to share the same IP address without conflicts. The port number configuration is determined by the software you are using:

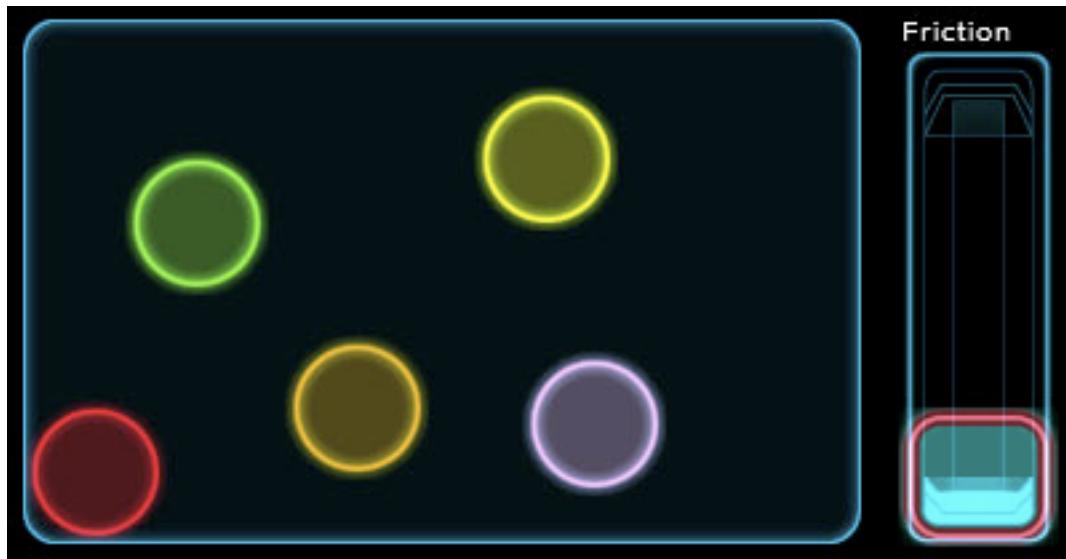
- **Port 8001 and 8002 should not be used**, because the JazzEditor and JazzDaemon use those ports to establish connections with the Lemur. If you try to use another application while the JazzEditor is open, the application will not be able to access ports 8001=8002. Similarly, if you launch the JazzEditor while another application is using ports 8001-8002, it will be unable to connect to the Lemur.
- For Cycling 74's **Max/MSP**, the port number can be anything. **8000** is the typical value used.
- For Native Instruments' **Reaktor**, the default port number is 10000.
- For other OSC-compatible software or hardware, consult the Target's documentation for details on port settings.

Once you have loaded a Project from the JazzEditor onto the Lemur (which is automatically done on Connect), or open a Project from the Lemur's internal flash memory, the OSC messages are handled by the Lemur. This means you can close the JazzEditor and let the Lemur communicate with OSC Targets.

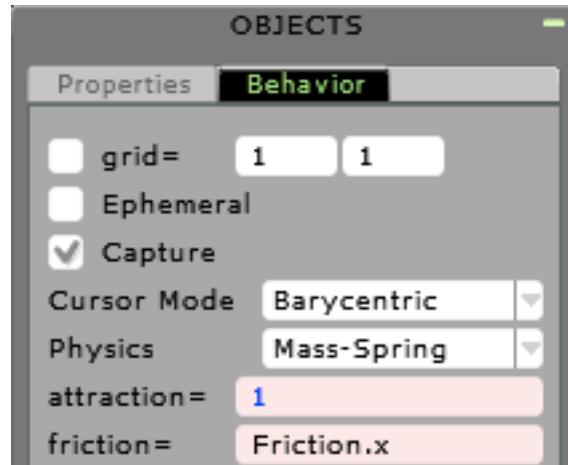
# Chapter 9 - Going further with the JazzEditor

## 9.1 Control your Objects with your Objects

One of the many exciting features of the Lemur is the ability to control the behaviour of your Objects via other Objects on the Interface. Let's go through a basic example: create a Multiball and a Fader, and name the Fader Friction. Check the Fader's Label flag in its Properties tab to make things clearer.

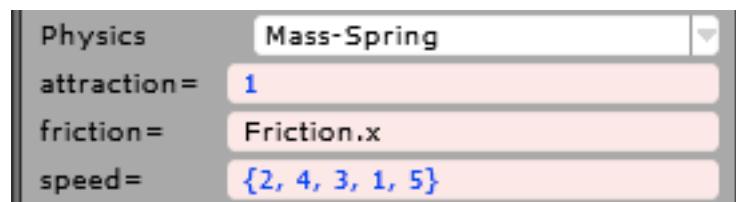


Now switch to the Behavior tab of the MultiBall Object and change the Physics parameter to **Mass-Spring**. Now type in **Friction.x** (the Fader's x Variable) as the value for friction:



Now you control the MultiBall's friction parameter via the Fader, it's as simple as that. Use any value of any of the Objects dwelling on your Lemur Interface to control any of the parameters. This can lead to very interactive Interfaces...

If you have sub-objects you can address their behaviour properties individually by using a **vector** or array instead of a singleton. If you have five balls in a MultiBall Object and use a vector with 5 components to denote, say, **speed**, each ball will at its own speed value. Of course this also works with attraction, friction or any other parameter of Objects with multiple sub-objects.



You could have the Fader display the chosen friction value simply by checking its **value** flag. But sometimes you want to have the value displayed at a different place on your interface. You can even display values on top of Objects if you wish, and that's what we're going to do now. Create a Monitor Object and position it right in the middle of the MultiBall area.

Name the Object *Grip*, check the **Label**, **Transparent** and **value** flags. Fill in **Friction.x** for value because we want to display the friction parameter set via our Fader.



Now set the **Font** size of the Monitor to **13pt** and choose a nice, contrasting colour. You should get something like this:



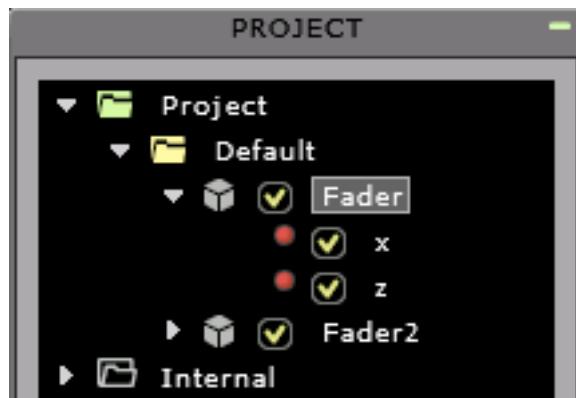
If you move the Friction Fader, its value is displayed via the Grip Monitor. You can still use the MultiBall Object, as if the Monitor wouldn't exist, because the Monitor is transparent to touch. Don't mix this feature up with the **Transparent** flag of the Monitor which is only making the background of the Monitor invisible. You can always "reach through" a Monitor Object, even if its Transparent flag is unchecked. This can come handy if you want to hide Objects on your Interface but still use them with your fingers.

## 9.2 Making your own Variables

Now, let's say you have that special Reaktor patch with a custom parameter having an extra wide range. You want fine-grained control with a Fader in the lower range but you also need the possibility to control it over the complete range. How would you do that?

By creating two Fader Objects, and use second as a multiplier for the Value of the first one through a User-defined Variable that gets transmitted via OSC and MIDI.

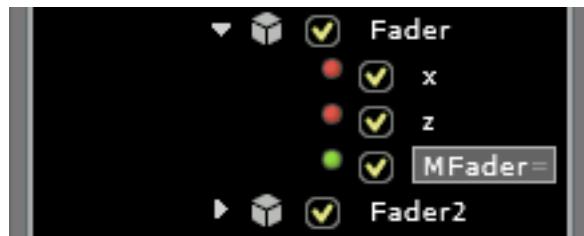
So let's create two Faders, and select the first one in the Project panel:



Now click on the **Create Expression** button below the Project panel.



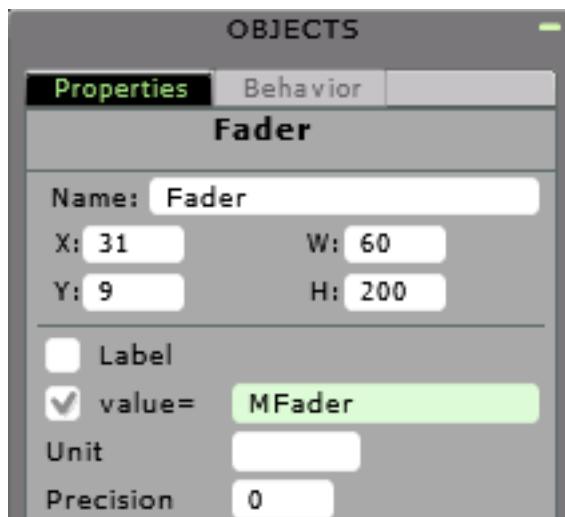
Type in the name **MFader** into the dialog and click OK. You should see the Variable named **Mfader** in the tree, local to the Fader.



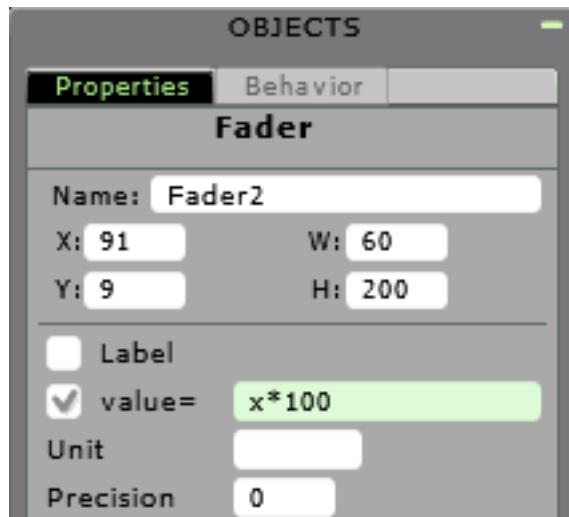
We want to scale the output of the first Fader with the value of the second. Do this by clicking on the fresh Variable in the Project panel. Type **x\*(Fader2.x\*100)** into the Script field of the Variable. Make sure that the checkbox in front of the local Variable is checked. This ensures that the Variable is actually transmitted via OSC and MIDI. It should look like this:

You might wonder why we addressed the Fader's value with **x** directly and not via **Fader.x**. This is possible because the Variable we defined is local to the Fader so that it "knows" the address of its Parent. If we had created the User-defined Variable at the Project level, the full address (**Fader.x**) would be required.

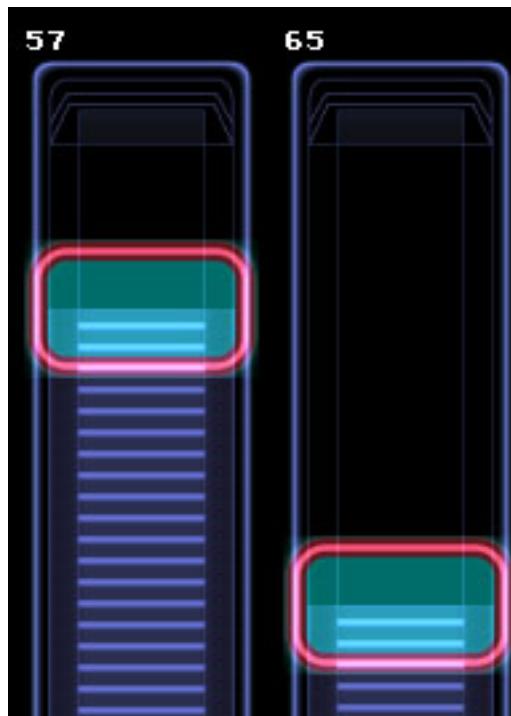
Of course you want to see the output of the Variable. Open the Properties tab of the first Fader and check the **value** flag. Type **MFader** into the **value** field. As we don't care about decimal places this time, set the **Precision** to 0.



For the second Fader also check the value checkbox and type  $x*100$  into the value field. **Precision** can also be set to 0.



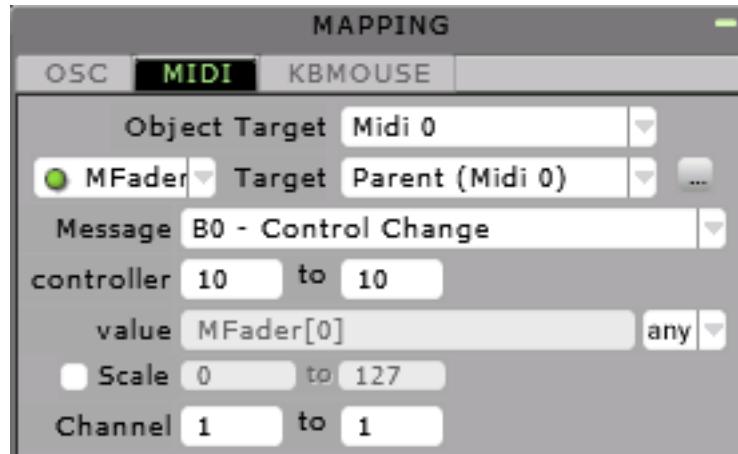
If you now move the Faders you will see the relevant values displayed above. Move the second Fader and the value of the Variable will change, because it gets multiplied by a factor between 0 and 100 depending on the value of the second Fader ( $Fader2.x*100$ ).



Please note that the displayed values are not those actually produced by the Faders. They both still produce values between 0 and 1. Those values are combined in the calculation of the local Variable **MFader** that is displayed above the first Fader. The factor that's used for multiplication is displayed above Fader 2.

Now, what if you want to multiply with multiples of 10, only? The Fader has **grid** parameter for quantization of the output.

We want to transmit the local Variable *MFader* via MIDI. This is easily done via the MIDI panel of the first Fader. The Variables menu contains the Variable *MFader* and we can choose it to be transmitted to MIDI Target 0 (or any other Target we want). Choose **Control Change** as Message and **Controller 10** as controller. Make sure that you uncheck the **Scale** checkbox, as we want the values to be transmitted as they are and not extrapolated to values between 0 and 127.



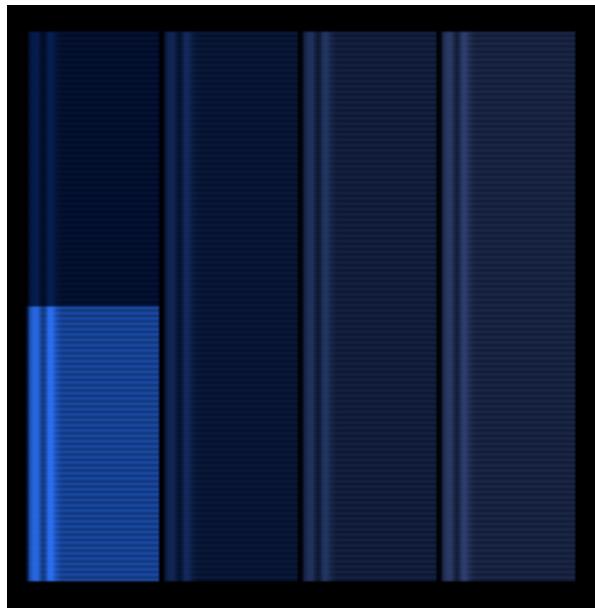
Now your Lemur will output values between 0 and 100 to MIDI Target 0 depending on the state of the two Faders.

### 9.3 Using Vector Variables

The Lemur doesn't only know data structures containing single values. There are also vector/list variables that consist of more than one value. Vectors Variables are used by the Breakpoint, Leds, MultiSlider, MultiBall, Pads, and Switches Objects.

This can be convenient when you need controls for ADSR envelopes or other tightly interrelated groups of parameters. Let's use a MultiSlider to produce values for an envelope working somewhere in a Synthesizer and triggered with a Pads Object:

- Create a **Pads** Object and name it **Trig**, then create a MultiSlider and name it **Env**.
- Click the MultiSlider's Properties tab, and set the number of sliders to 4. Move and resize the MultiSlider so you can control each slider easily. Maybe something like this or a little bigger?



We're going to use the first slider for **Attack**, the second for **Decay**, the third for **Sustain**, and the fourth for **Release**. As with the x variables of all Lemur objects, the MultiSlider x variable ranges between 0 and 1. This is not really enough of a range for the time values of our envelope, so we need to scale these values. We'll also need to create three Expressions, one for each of the envelope's time values.

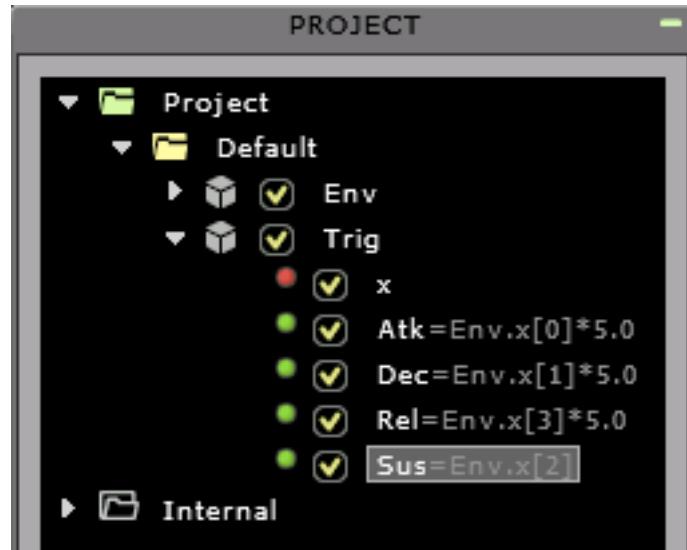
- Click on the Trig Object (Pads) in the Project panel so that its name is selected.
- Click the Create Expression button to create a new User-defined Variable.
- Name the expression **Atk** and press return
- In the corresponding script field enter the Expression shown below:

```
= Env.x[0] *5.0
```

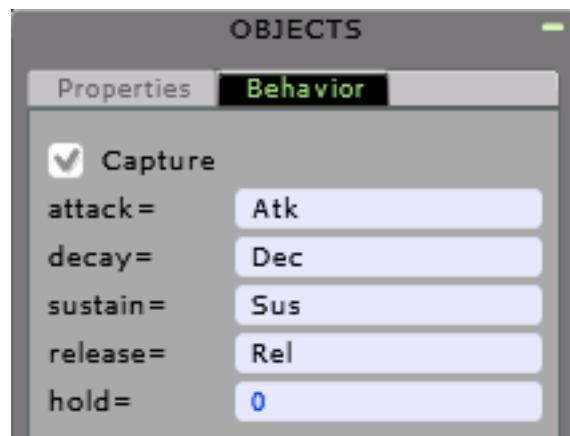
This means that the Atk Variable will use the value of the first slider (with the index of 0) of the MultiSlider we just created.

Create similar Expressions for **Dec** (decay), which will use **Env.x[1] \* 5.0**, **Sus** (sustain), which will use **Env.x[2]**, and **Rel** (release), which will use **Env.x[3] \* 5.0**. Note that we do not scale the value for Sustain, since this is just a value between 0 and 1.

When you're done, your Project panel should look like this:



The final step is to enter these Variable names in the Behavior Properties of the *Trig* Object. Enter **Atk** for Attack, **Dec** for Decay, **Sus** for Sustain, and **Rel** for Release as shown below. As the Variables are local to the *Trig* Object, you don't have to write out the complete addresses (which would be *Trig.Atk*, *Trig.Dec*, etc.).



Now you can test your envelope by hitting the Pads Object. You don't even need an attached Synthesizer to see what's going on. Dial in the envelope values via the MultiSlider and you will see how the changes of the sliders affect the brightness envelope of the pad.

## 9.4 Using Custom MIDI Messages

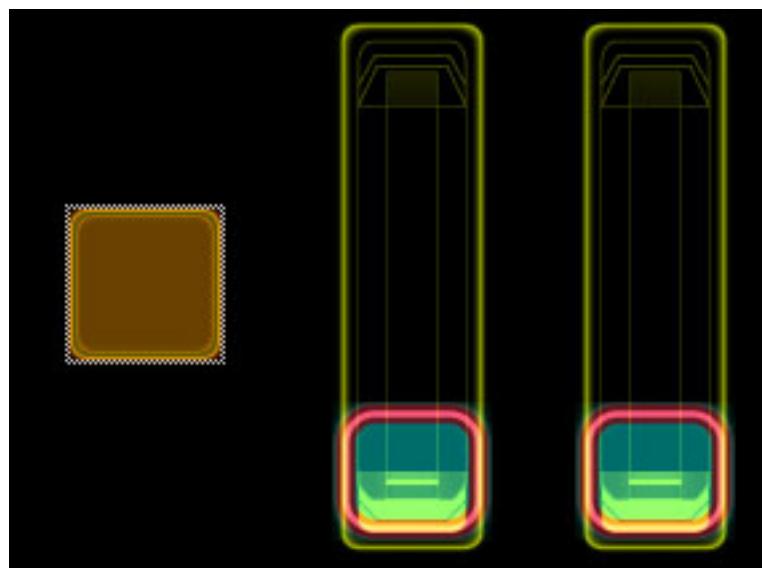
Custom MIDI messages provide simple means to generate very complex MIDI data by entering Expressions for all the parameters of the chosen MIDI message. When using the Mapping panel's MIDI tab, you can only assign the selected Variable to one parameter of the message. With Custom MIDI messages you gain complete freedom to assign any value or expression to any one of the different parameters of the message. And remember that expressions can use all values of all Objects in your Project.

For a Note On message the parameters would be:

- Pitch
- Velocity
- MIDI Channel

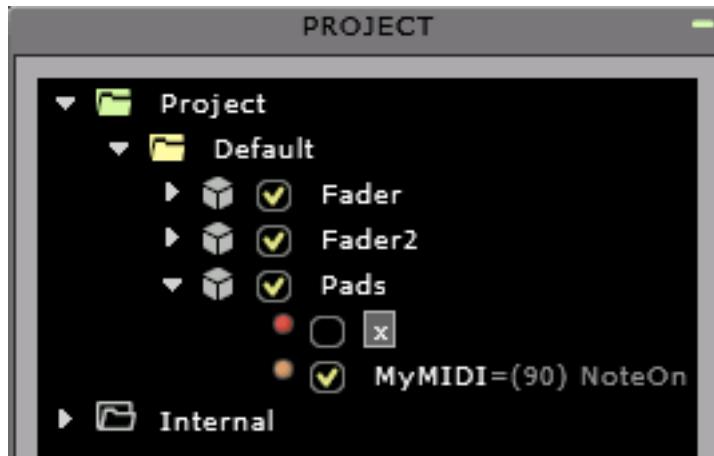
You can even use an additional expression to control the triggering of the MIDI message. A **Trigger Mode** menu is also available for each of the Custom MIDI's parameters. Please read Chapter 7.3 for details on Trigger modes.

As a small example, let's create a Pads Object and two Faders, then select the Pads Object in the Project panel.

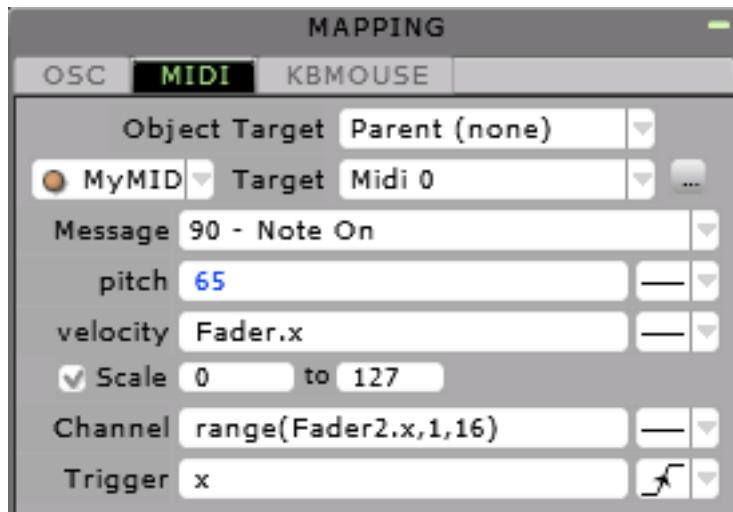


Next, click on the **Create Custom MIDI** icon below the Project Panel to create a customized MIDI message local to the Pads Object. and call it **MyMIDI**.





Now look at the MIDI panel of the Custom Midi message and choose **Note On** from the message menu.



We want a fixed pitch for the generated note messages and set the pitch field to **65**.

For **velocity** we use the value of the first Fader (Fader.x), scaled to the MIDI range of 0 to 127 via the **Scale** fields. Note that we've set the corresponding Trigger modes to "none" (change of value of the Fader doesn't trigger message sending).

The **Trigger** field is set to **x**, which is the state of the Pad. By setting the corresponding Trigger mode field to "**up**", the MIDI message is sent when x leaves 0, i.e when the Pad is pressed.

Now we can trigger notes with the Pad and control the velocity with the first Fader. As a playful addition we assign the other Fader's value to the MIDI channel. Here we use range function to get values between 1 and 16.

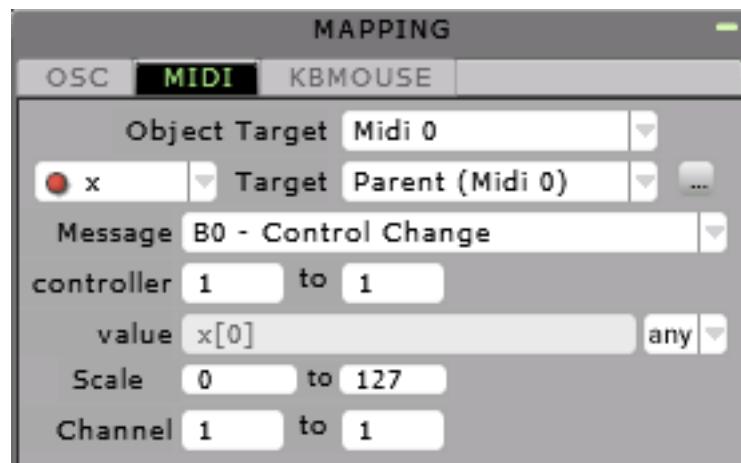
Now you can choose the MIDI Channel for the outgoing note messages. This might be an exotic application but it's fun – stack up a pile of sounds in multimode in your favourite synth and trigger different sounds by moving the channel fader on your Lemur. Pay attention to the different Trigger Modes we use. Pitch, Velocity and Channel are set to None. Only a change in the Pads.x value triggers transmission of the MIDI message.

It's important to note that a Variable assigned to a Custom MIDI can output data but will not react to the corresponding incoming message. Custom MIDI messages are **not bidirectional**.

## 9.5 Bidirectional Control

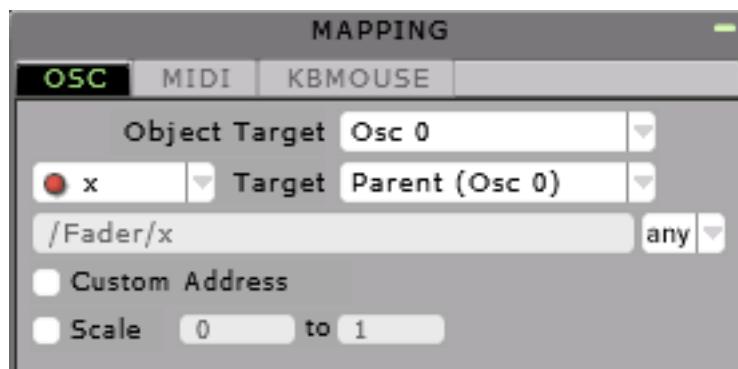
Not only can the Variables of a Lemur Project output data, but they can also be controlled by external software or hardware via MIDI or OSC. This is what we refer to as **bidirectional control**.

If you want to control a Variable via MIDI, just set up the correct MIDI port for a Target on your Lemur JazzDaemon Target settings. Then choose that MIDI Target for the Variable on the MIDI panel.



If you have, for example, a Fader's x value set up for transmission of MIDI controller 1 on MIDI Target 0, it will also be controllable via incoming controller 1 data on the input port chosen for that MIDI Target.

Similarly, you can control all values via OSC. The different values are controlled via their OSC addresses you already know.



For controlling a Fader via OSC, just set up the OSC Target for the x value on the OSC panel and produce the OSC messages on the Target side. In the Fader's case you would send values between 0 and 1 to the OSC address /Fader/x to the current IP and port of the Lemur. If you've set up a custom OSC range for Fader.x, it will respond to values inside that range.

Please make sure that you think about the **hierarchy** of Objects on the Lemur. If the Fader is inside of a Container the address would be **/Container/Fader/x**.

Note that it's also possible to switch Interfaces via OSC. Just send an OSC message in the format "/interface InterfaceName" to the Lemur's IP (port 8000). In our case this would look like "/interface 1 Demo" for switching to the first one. For names that include spaces there is an additional consideration to be made. The OSC software has to send the name as a single string. This might involve putting it in quotes in the software. This is the case for Max/MSP.

For more about working with OSC, MIDI and bi-directional control please have a look at the extensive workshop chapters we offer on our website. There you'll learn everything about how the Lemur loves Max/MSP, Reaktor and other OSC enabled applications.

## 9.6 Defining and Using Functions

In addition to built-in functions and operators, you may define your own mathematical **functions** to use in User-defined Variables and expressions. Here's how to do it:

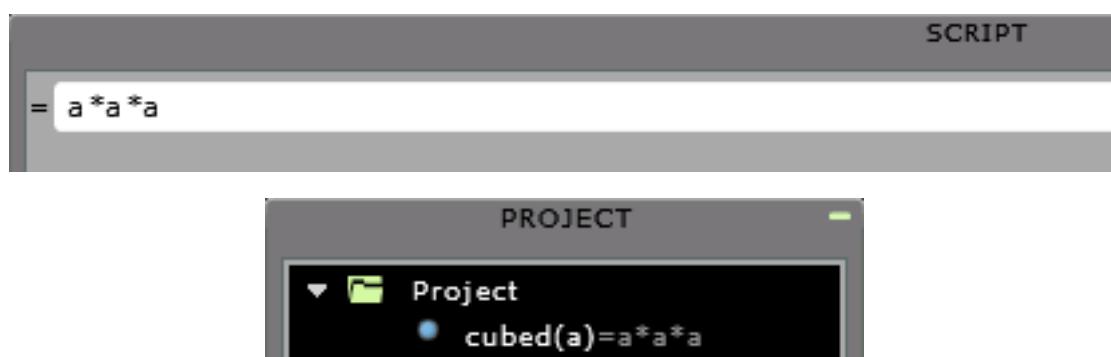
Select your Project's folder in the Project panel, and click the Create Expression button to create a new Expression.

The Expression is created at the level of the hierarchy you have currently selected. If you have an Object selected it will be an Expression local to that Object. If you have the Project folder selected the Expression will go there and be available for use anywhere in the Project.

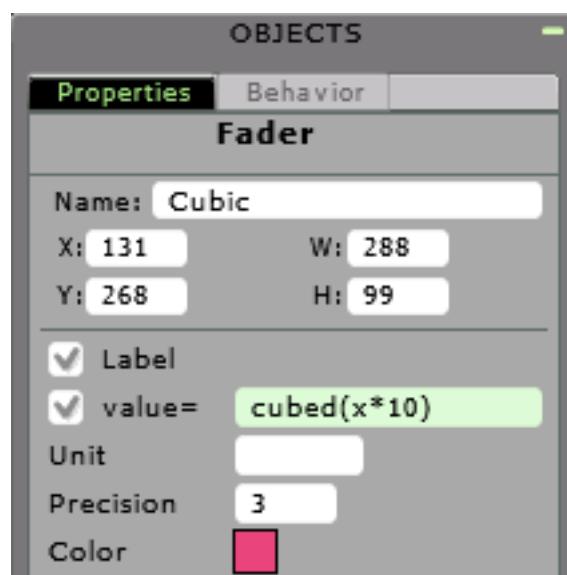
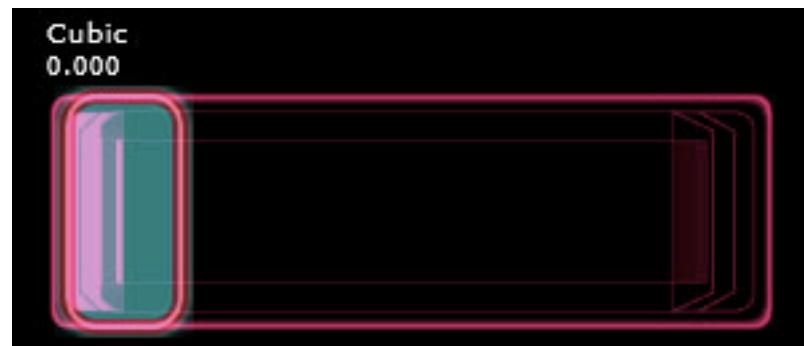
- Type the name of the function and its arguments in parentheses. For example, to define a function named *cubed* that takes one argument, you would type:



- Click OK and go to the Script panel to type in the definition of your function as an expression. You can reference other variables if needed. The example below just multiplies the function argument *a* by itself twice to raise the input to the third power.



How to use the new function? Let's try it out on a simple Fader: create a Fader Object, make it horizontal for a change (by stretching it horizontally) name it **Cubic** and check both its **label** and **value** checkboxes in its properties tab.



Set the value field to **cubed(x\*10)**. You are now using the Function we just created to raise the x value (which is multiplied by 10 beforehand) to the power of three. The variable argument of the Function is replaced by the Expression  $x*10$ .



Move the Fader and you will see that it displays an exponential range from 0 to 1000.

# Chapter 10 - Introducing Multi-line Scripts

Multi-line scripting is one of the most powerful features of the JazzEditor. It can be used, in conjunction with a set of powerful built-in functions and operators, to instruct your Lemur to perform specific tasks at specific times. From simple actions such as changing a MultiSlider color when a switch is pressed, to complex actions such as commanding a Multiball's balls to follow set trajectories... possibilities are truly endless.

## 10.1 Creating a Script

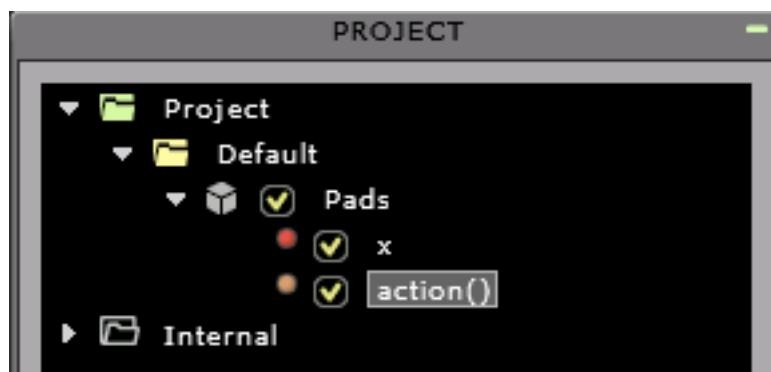
To create a new script, the first step is to select the element you want to associate the script with in the Project panel, be it the entire project, a specific interface, container or object, and click on the **Create Script** icon.



A dialog window will appear, asking you to name the script appropriately. Avoid names taken by internal functions, such as *set*, *firstof*, *replace* and so on, as your script would then temporarily become a substitute for that particular function. You can refer to the Internal folder in the Project panel for a quick look at the built-in functions list.



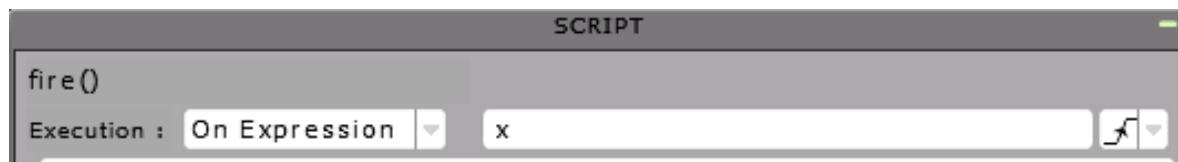
When created, your script will appear in the Project panel, preceded by an orange dot, and followed by brackets containing the script's arguments, if any, representing the values passed to the script at the time of call.



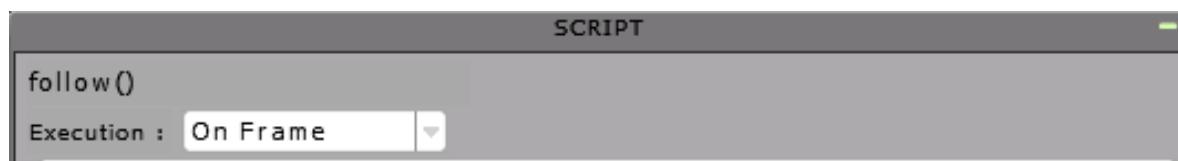
## 10.2 Execution

Before beginning to encode a multi-line script, it is essential to define precisely when the Lemur should perform the desired task and set the script's execution mode accordingly. To this end, a choice of 6 different **Execution** modes is available to cover the range of possible situations:

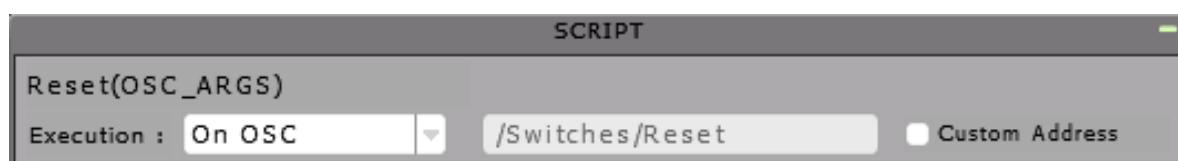
- **On Expression:** the script's execution is triggered when the value of the variable or expression entered in the adjacent field changes in the manner specified by the **Trigger menu**. This is the same Trigger menu that we find in the MIDI, OSC or KbMouse tabs when we assign OSC, MIDI messages or Keyboard and Mouse commands to objects. The Trigger menu offers several options: the script can be triggered whenever the variable or expression changes, when it rises from 0, when it reaches 0, when it reaches or rises from 0, each time it increases above its previous value, or each time it decreases below its previous value.



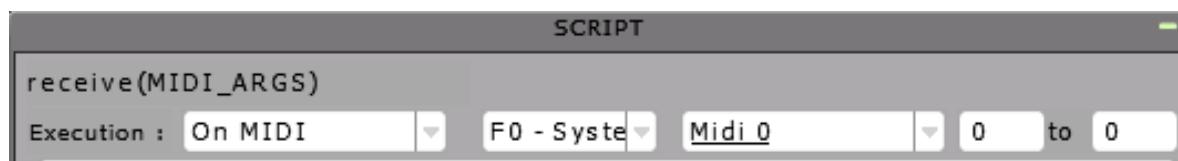
- **On Frame:** the script's execution is triggered each Lemur frame, that is, each time the Lemur scans an interface to recalculate the states of the objects on display. The rate at which the Lemur scans a page is inversely proportional to the amount of elements to be calculated. With a maximum rate of 60 frames per second for a blank page, this rate slows down with increasingly complex interfaces.



- **On OSC:** the script's execution is triggered upon reception of the OSC message displayed in the adjacent field, the script's arguments (OSC\_ARGS). By default, this is set to the script's name (*Container/Fader/Scriptname*), but can be customized by ticking the **Custom Address** checkbox.



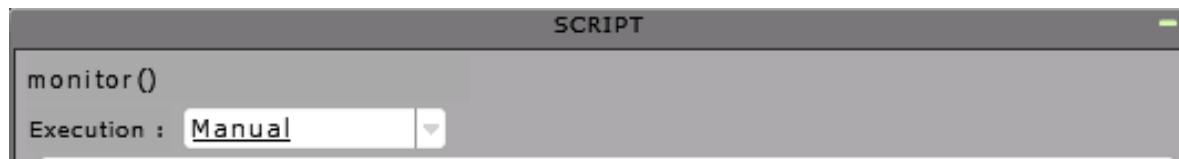
- **On MIDI:** the script's execution is triggered upon reception of the MIDI message (MIDI\_ARGS) defined in the adjacent fields: message type, MIDI Target, MIDI channel.



- **On Load:** the script's execution is triggered when the project is loaded on the Lemur.



- **Manual:** the script's execution is triggered when called up in another script or expression.



## 10.3 Variables

All the variables contained in your Project, built-in or User-defined Variables alike can be recalled in your code and combined with functions and operators to form a script that your Lemur will understand. In addition to these Project Variables, the Multi-line Script panel allows you to declare (create) variables specific to your Script, allowing you to store values while the script is in use. These “declared” Variables are only visible inside the Script panel (they do not appear in the Project panel), and expire when the script is not in use.

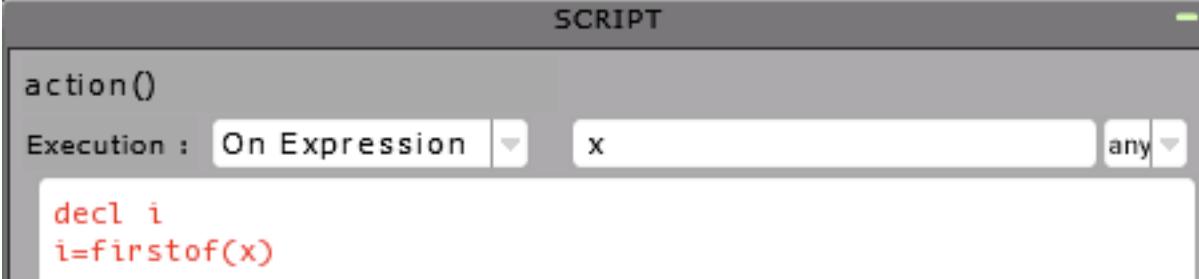
To declare a script Variable, use the statement **decl**:



A script Variable can be declared and equated to an expression in a single line too:

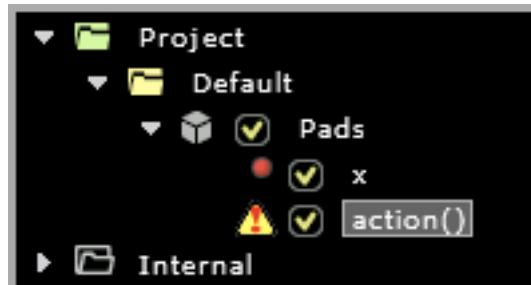


**Note that the Multi-line Script syntax requires that each line of code ends with a semicolon (;).** If any error, such as forgetting a semicolon, is present in the script, the lines of code will become red, and a “caution” triangle appears in the Project panel next to the script.



```
action()
Execution : On Expression x any
```

```
decl i
i=firstof(x)
```



## 10.4 Attributes

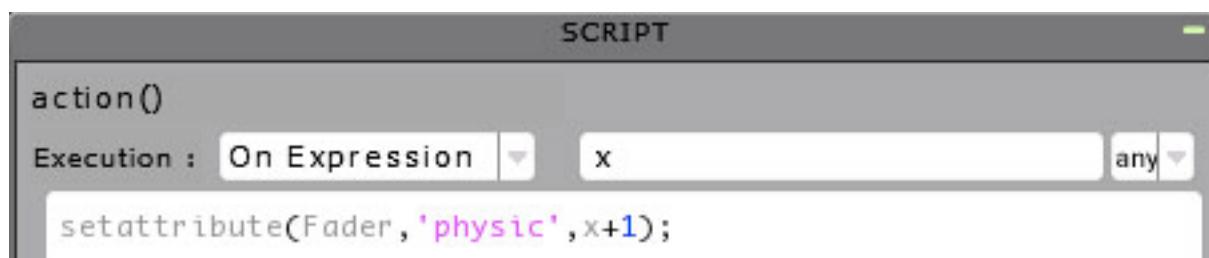
The Objects' attributes include all those Object parameters which are editable in real-time, either through instructions in a script with the function **setattr(object,'name', value)**, or directly through an OSC command in the form **/Container/Fader @grid 1** (activate the Fader's Grid parameter), **/Container/Fader @rect 300 100 60 200** (change the object's position to X=300 and Y=100, and the object's dimension to W=60 and H=200), or **/Container/Text @content "Hello !!!"** (change the displayed text).

Let's look at the Fader's attributes for instance. The table below presents a list of all its attributes, giving the possible values these attributes can take, together with a brief description. A list of all the attributes for each Object can be found in the Object Reference Chapter (Chapter 12).

### The Fader Object's attributes

Attribute	Value	Parameter
capture	0 or 1	capture off/on
color	integer*	color of the object
cursor	0 to 3	cursor mode
grid	0 or 1	grid off/on
grid_steps	1 to 33	number of grid steps
label	0 or 1	label off/on
physic	0 to 2	physics mode
precision	0 to 6	decimal places of displayed value
rect	{X,Y,W,H}	object's position and dimensions
unit	text	unit of displayed value
value	0 or 1	value off/on
zoom	1 to 50	zoom in object

Imagine, for instance, that you would like to change your Fader's physic mode from Interpolate to Mass-Spring while you're playing, by pressing a Switch on your interface. Your script, created locally to your Switch object would consist of the following line:



With this line, we instruct the Lemur to set the 'physic' attribute of Fader Object to the value  $x+1$ ,  $x$  being the  $x$  Variable of the switch since the Script was created local to the Switches Object. This means that the value of the 'physic' attribute will be equal to 1 when the Switch is off ( $x=0$ ), and to 2 when the switch is on ( $x=1$ ). Note that the Execution mode is set to "On expression  $x$ ", with Trigger mode "any", so that the script is executed each time the switch is pressed on or off.

By the way, it is useful to know that when you are working in the JazzEditor, the function **getattributelist(object)** can be used to return a list of attributes for the chosen object in a Monitor object on your Lemur

## 10.5 Built-in Functions and Operators

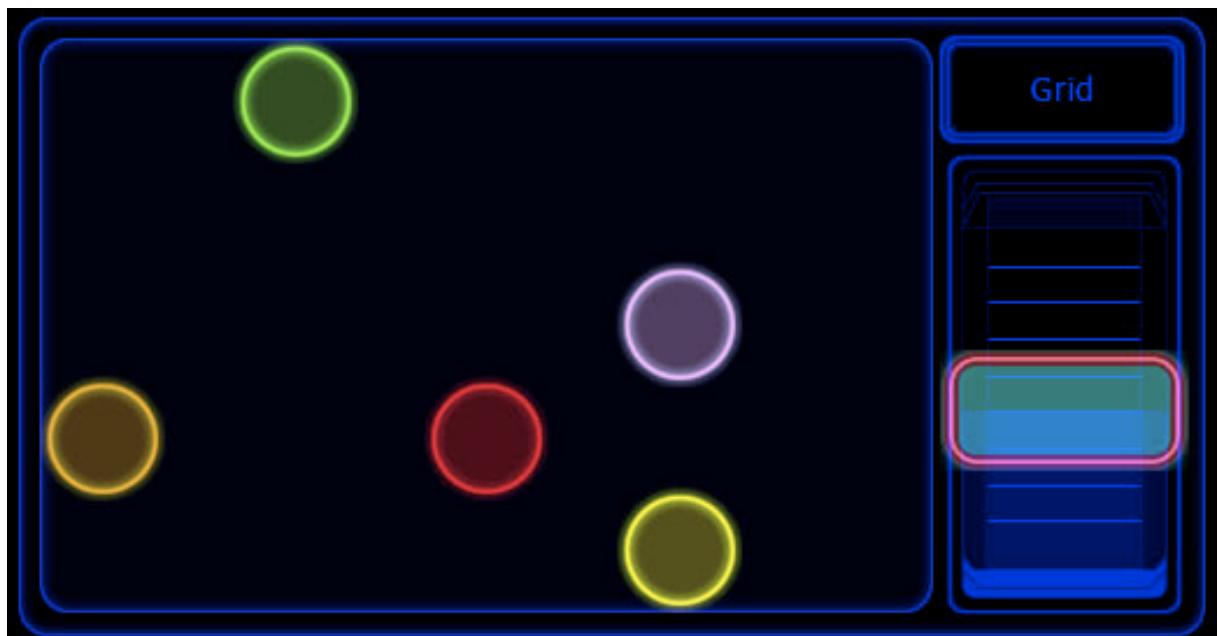
The JazzEditor features a wide range of handy built-in functions, all listed in the Project panel's **Internal** folder, whose purposes are to make your life easier when defining a single-line expression for a User-defined Variable or when encoding a multi-line script. Some of these functions, such as the **firstof(x)** function (returns the position of the first non-null item in a vector) can be used in both single-line expressions and multi-line scripts, while others, like the **setattribute(object,'name', value)** mentioned previously, are specific to multi-line scripting.

In addition, the multi-line Script panel also understands a wide range of operators such as the **arithmetic**, **comparison** and **logic** operators, but also operators denoting assignment or bitwise operations. Basically assignment operators allow the same variable name to contain different values at different times during the script execution, while bitwise operators allow operations on array patterns at the level of their individual elements.

Please refer to the Parser reference on Chapter 13 for descriptions of all built-in functions and operators.

## 10.6 Examples

We have a Multiball, together with a CustomButton and a Fader. We'd like to be able to switch the Multiball's grid on or off with the CustomButton, and set the Multiball's number of balls with the Fader.



Let's take care of the Multiball's grid first. In parser terms, switching a Multiball's grid on or off means setting its 'grid' attribute to 0 (off) or 1 (on).

For the script to be executed when the CustomButton is pressed, we need to associate the Script with the CustomButton's x variable. To do this, we'll first select the CustomButton in the Project Browser and click on the Script icon. We'll name this script *gridon* and set its execution mode to **On Expression**. We then fill the field adjacent to On Expression with x, and keep the trigger menu to any: the script will be executed whenever the CustomButton's x variable (its on or off state) changes. As far as the code is concerned, we'll use the **setattr(object,name,value)** function, where *object* will be Multiball, *name* the attribute's name 'grid' and *value* will be x, the CustomButton's state:



Let's take care of the Fader next. Again we'll use the **setattr(object,name,value)** function, but this time the attribute's name will be 'nbr': the number of balls.

To make this practical, we'll use a 10 steps grid for the Fader, so that each step upward adds a ball, and each step downward takes one off. We can translate this into a number with an expression local to the Fader: select the Fader in the Project Browser, click on the Create expression icon, name it num for instance and equate it to the expression **round(x\*9)+1**. That way num will take integer values between 1 and 10, values that we'll use in our script with the **setattr(object,name,value)** function to set the number of balls. We'll associate the script with the Fader object, to be executed **On Expression num** (each time the Fader modifies the number of balls) as follows:



An alternative would be to replace the Fader object with two pads: one for incrementing the number of balls, the other for decrementing. To make it more interesting, we can also add a MultiSlider for setting the Multiball's color, and a Fader to zoom in the Multiball for increased precision.



We'll begin with the + Pad together with a script we'll call *increment*. Again the execution mode will be **On Expression x**, but this time we'll set the corresponding trigger menu so that the script gets executed only when x goes from 0 to positive (we only want the script to be triggered when the pad is pressed, not when it's released).

Our script begins by declaring a new local variable **b**, which will correspond to the current number of balls, i.e. the value that we'll increment. We can use the function **getattribute(object,name)** to get this info. We'll then use the arithmetic operator **++** (which simply means "increment") to increment our **b** variable before using it with the **setattribute(Multiball,'nbr',b)** function.



Similarly, we'll create a second pad together with a script called *decrement*. The difference here will be the use of the arithmetic operator **--** (which means "decrement").

```
decrement()
Execution : On Expression x
decl b;
b = getattribute(Multiball, 'nbr');
--b;
setattribute(Multiball, 'nbr', b);
```

The MultiSlider object will be used to set the Multiball's color, with three sliders for mixing the primary colors (Red, Green and Blue) according to the RGB additive color model. In the script, we'll use the **setattribute(object,name,value)** function to set the Multiball's 'color' attribute with a value computed by the **RGB(r,g,b)** function, which will convert the 3 sliders (each between 0 and 1) into a single color value that the parser understands. Note that the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127.

```
changeColor()
Execution : On Expression x any
setattribute(Multiball, 'color', RGB(x[0], x[1], x[2]));
```

Finally we'll create a script for the Fader object in order to control the Multiball's **'zoom'** attribute. Since the 'zoom' attribute takes values between 1 and 50, we'll need to scale the Fader's x variable by multiplying it by 50 and adding 1:

```
zoom()
Execution : On Expression x any
setattribute(Multiball, 'zoom', (x*50)+1);
```

The **setattribute(object,name,value)** can also be used to change the dimensions of an object and/or move it around by changing its **'rect'** attribute. The 'rect' attribute takes values in the form {X,Y,W,H}, where, as in the JazzEditor's Properties tab, X is the horizontal position of the object, Y the vertical position, W the width of the object and H the height.

In this last example, we've got large Container called *Containerbig* filled with Multiball objects, and in order to gain space, we've enclosed it within a smaller Container called *Containersmall* and add a Fader to scroll up and down the Multiballs.



The script, local to the Fader object, and whose execution is initiated by any change in its x variable (each time the Fader moves up or down), uses the value of x to multiply the Y element of *Containerbig*'s rect attribute, thereby changing its vertical position within the smaller container:

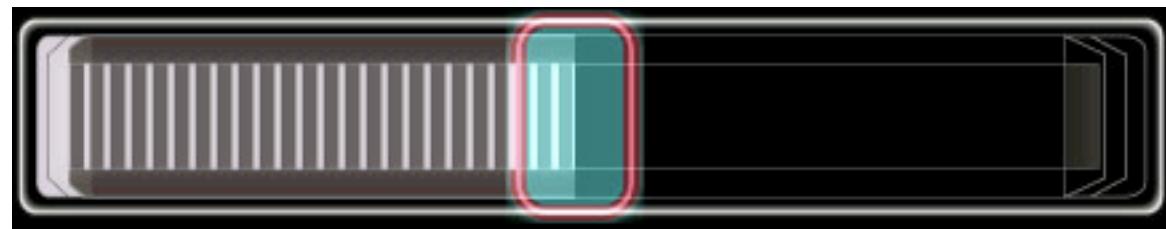
```

SCRIPT
move()
Execution : On Expression x any
setattribute(Containersmall.Containerbig, "rect", {0,-300+(x*300),371,600});

```

Lastly, we can add an horizontal Fader object called Fader2 whose width is recalculated as the Fader is displaced. Maybe not the kind of Fader you'd like to use for controlling your master volume in a Live situation, but interesting nonetheless. Try changing the Fader's Physics to Mass-Spring to make it more fun.

Here we begin the script by declaring a new script Variable *temp*, then use the function **getobjectrect(object)** to obtain the existing value of the **rect** attribute in the form {X,Y,W,H}. In the third line of code, the Fader's new position modifies the third element of the vector *tmp* (*tmp[2]*), which corresponds to the width of the object. Finally the freshly calculated variable *tmp* is injected in the **setobjectrect(object,rect[])** to modify the width of the object.



SCRIPT

```
action()
Execution : On Expression x any
decl tmp;
tmp = getobjectrect(Fader2);
tmp[2] = 200 + 600*x;
setobjectrect(Fader2, tmp);
```

# Chapter 11 - Advanced Scripting

## 11.1 Conditional statements

- **if**

When the parser finds an if in the Script panel, it expects a boolean condition, that is, a logical proposition having one of two values: **true** (non-zero) or **false** (zero). If the condition is true, the parser executes the single line or block of code immediately after the condition, which, in the case of a block of code, will be enclosed within braces {}. If the condition is false, the statement in the single line or block of code is ignored.

The **if** structure in its simplest form:



(if **firstof(x)** is smaller or equal to 3, your Lemur sends a Control Change MIDI message on Target 0 for controller 20, with a value of 127, on MIDI channel 1.)

Several instructions can follow the boolean condition provided the block of code is enclosed within braces:



(if **firstof(x)** is smaller or equal to 3, your Lemur sends two Control Change MIDI messages: one for controller 20, one for controller 21.)

- **else**

It's also possible to make use of the **else** statement. If the condition is false, the execution jumps to the instruction immediately after else :

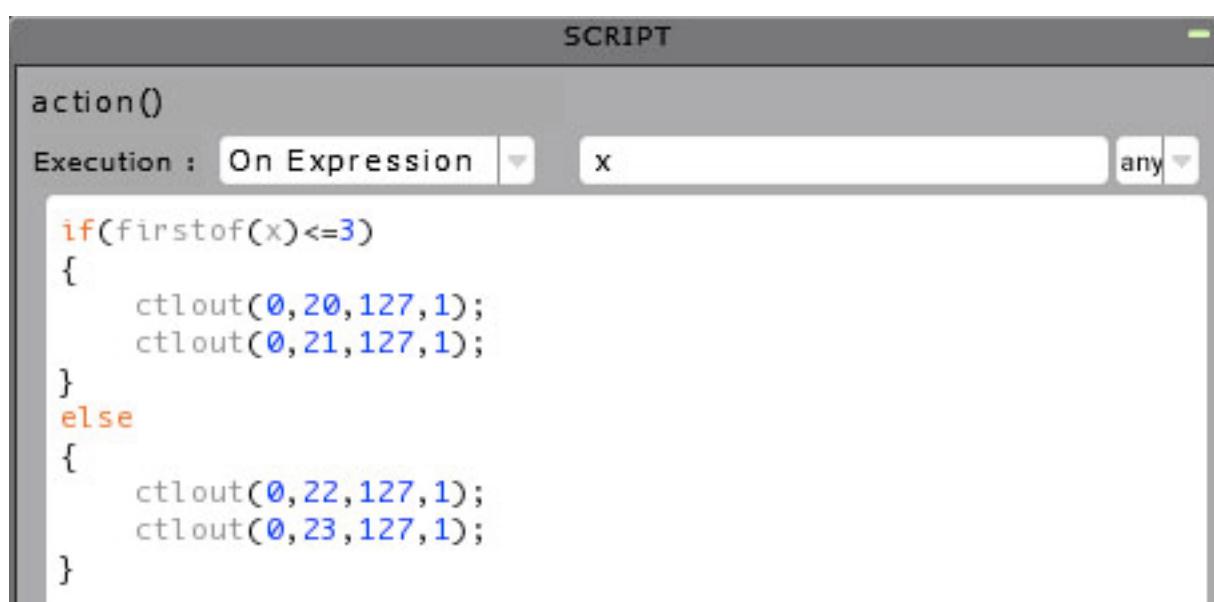


The screenshot shows the Lemur Script Editor window titled "SCRIPT". The code block contains:

```
action()
Execution : On Expression x any
if(firstof(x)<=3)ctlout(0,20,127,1);
else ctlout(0,23,127,1);
```

(if **firstof(x)** is smaller or equal to 3, your Lemur sends a Control Change MIDI message for controller 20. If **firstof(x)** is greater than 3, it sends a CC for controller 23.)

The else statement can also be a block of code contained within braces:



The screenshot shows the Lemur Script Editor window titled "SCRIPT". The code block contains:

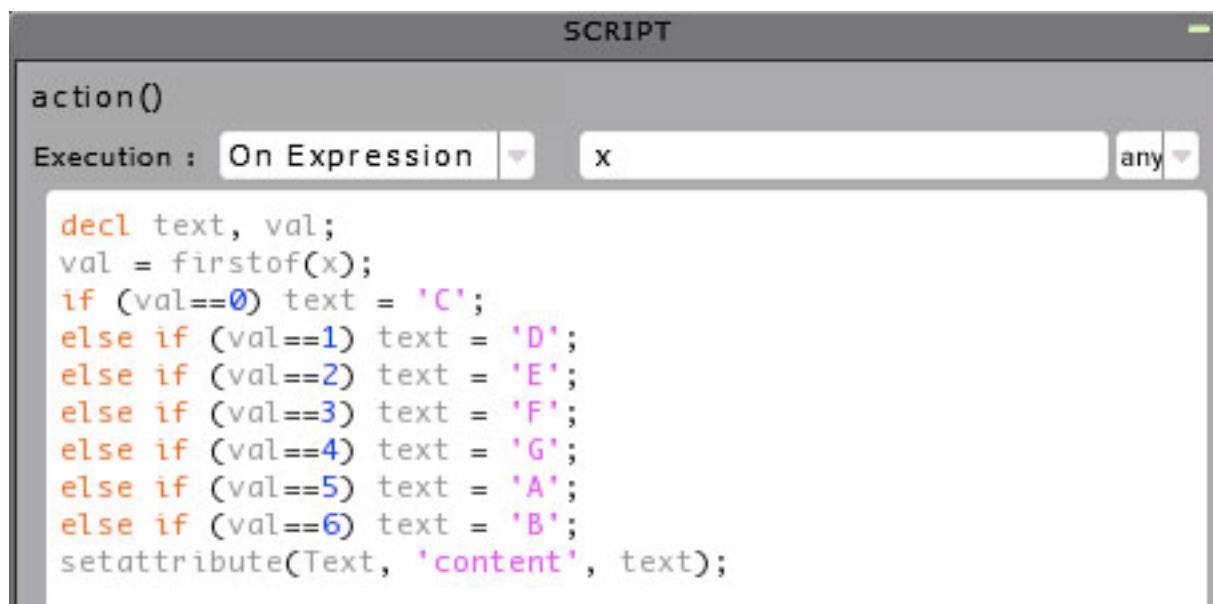
```
action()
Execution : On Expression x any
if(firstof(x)<=3)
{
    ctlout(0,20,127,1);
    ctlout(0,21,127,1);
}
else
{
    ctlout(0,22,127,1);
    ctlout(0,23,127,1);
}
```

(if **firstof(x)** is smaller or equal to 3, your Lemur sends two CC MIDI messages for controller 20 and 21. Else if it's greater than 3, your Lemur sends two CC MIDI messages for controller 22 and 23.)

- **else if**

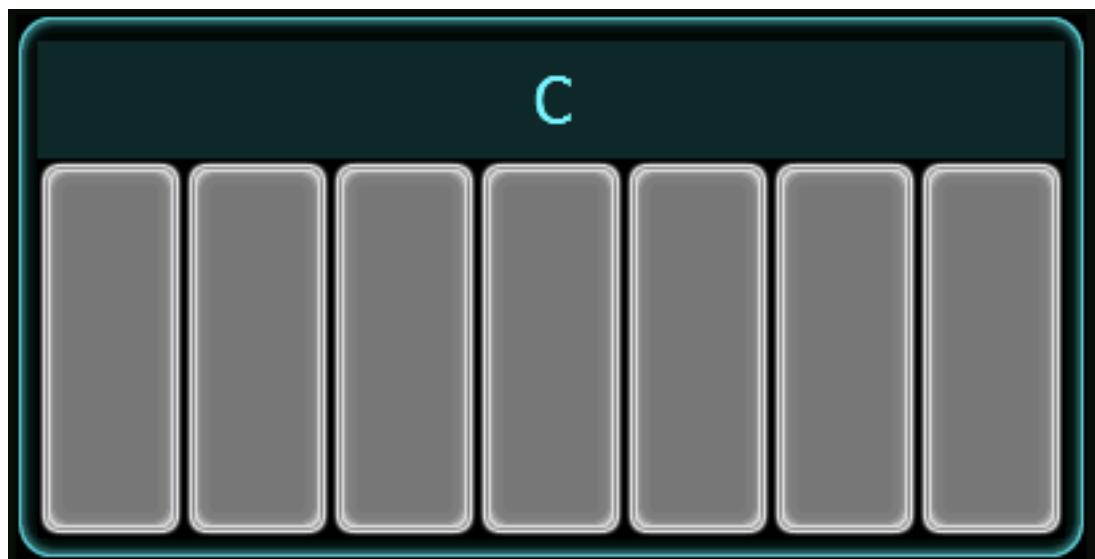
By using **else if**, it is possible to combine several conditions. Only the statement(s) following the first condition that is found to be true will be executed, and all other statement(s) will be skipped.

Let's go through an example. We're using a row of pads as a rudimentary keyboard, and we'd like a Text object to display the notes' names as we're playing: A,B,C... For this we'll use a script local to the Pads object, to be executed "on x" (each time a pad is pressed), and consisting of several **else if** statements. The **firstof(x)** function is used to return a value corresponding to the position of the pad being pressed, and the **setattribute()** function sets the Text Object's 'content' attribute according to the value of the **firstof(x)** function.



The screenshot shows a Max/MSP patch. At the top is a 'SCRIPT' object with the following code:

```
action()
Execution : On Expression x any
decl text, val;
val = firstof(x);
if (val==0) text = 'C';
else if (val==1) text = 'D';
else if (val==2) text = 'E';
else if (val==3) text = 'F';
else if (val==4) text = 'G';
else if (val==5) text = 'A';
else if (val==6) text = 'B';
setattribute(Text, 'content', text);
```



## 11.2 Loops

- **for**

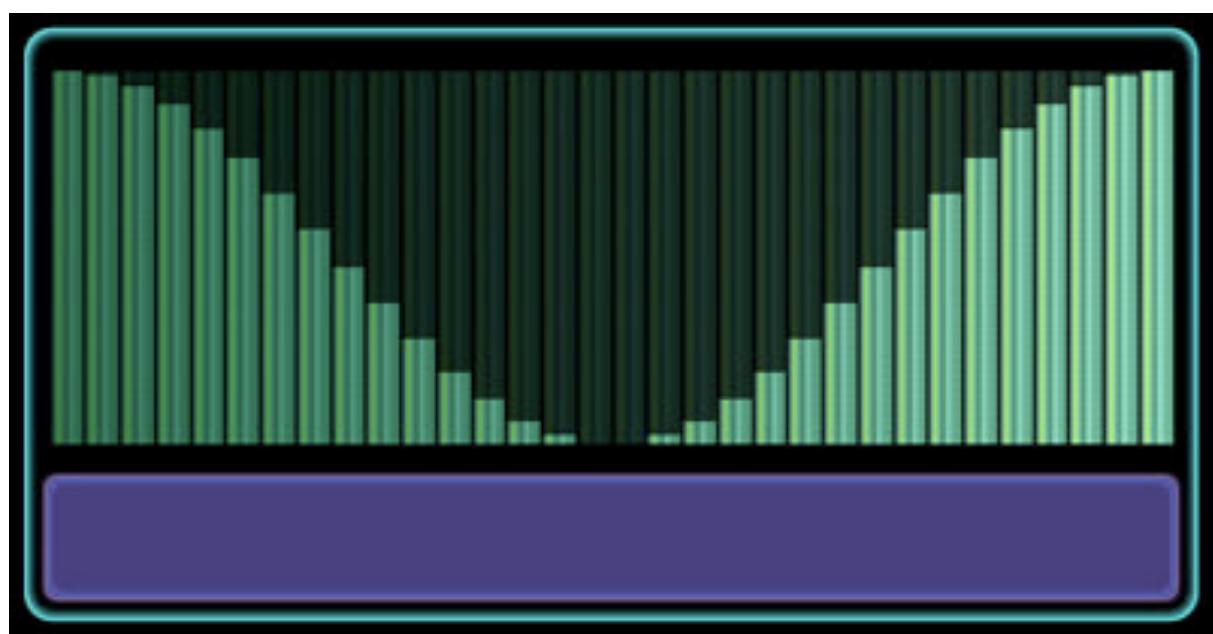
The **for** loop is an **iteration** statement which allows code to be repeatedly executed a specified number of times. The type of for loop used in the Script panel is characterized by three control expressions separated by semicolons: the **initializer** expression, the **loop test** expression and the **counting** expression. The initializer sets up the initial value of the variable within the looped block of code. The loop test expression is evaluated at the beginning of each iteration through the loop, and determines when the loop should exit. Finally, the counting expression is responsible for altering the loop variable.

Let's go through an example. In the script below, a **for** loop is used to set up the value of a MultiSlider's individual sliders on the basis of a cosine function. The initializer `i=0` sets up the initial value of the variable `i` to 0, which means that the first time, the loop block is executed for `MultiSlider.x[0]`, setting up the value of the first slider. The variable `i` is then incremented by the counting expression `++` to `i=1`, and the code within braces repeated for `MultiSlider.x[1]`. The variable will be incremented and the code repeated as long as `i < sizeof(MultiSlider.x)`, which means for each slider up to `MultiSlider.x[31]`.



The screenshot shows the Max/MSP Script panel titled "SCRIPT". It contains the following code:

```
action()
Execution : On Expression x any
decl i, size=sizeof(MultiSlider.x);
for(i=0;i<size;i++)
{
    MultiSlider.x[i]=0.5*(1+cos(i*2*pi/(size-1)));
}
```



- **break**

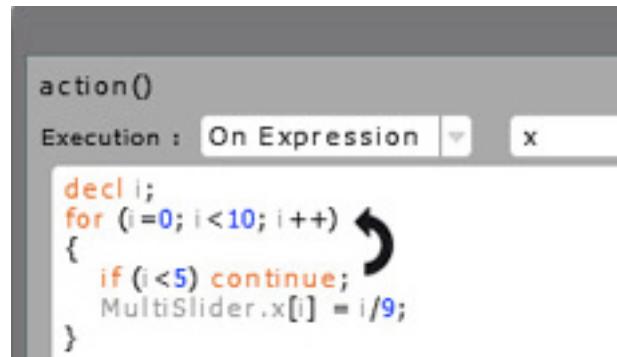
A break statement interrupts the execution of the current for/ while loop and jumps to the next statement.



```
action()
Execution : On Expression x
decl i = 0;
while (1)
{
    if (i<=4) MultiSlider.x[i] = i/9;
    else
    {
        MultiSlider.x[i] = 1;
        break;
    }
    i++;
}
```

- **continue**

The **continue** statement is used to **skip** the remainder of the loop body and continue with the next iteration of the loop. The effect is to prematurely terminate the innermost loop body and then resume as normal with the next iteration. If the iteration is the last one in the loop, the effect is to terminate the entire loop early.

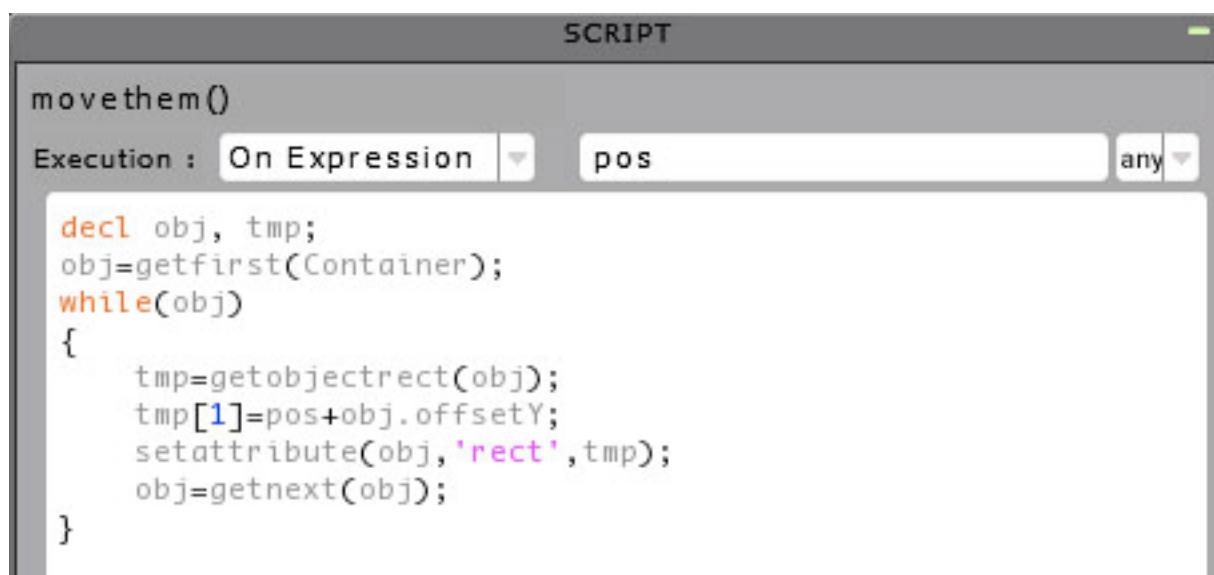


```
action()
Execution : On Expression x
decl i;
for (i=0; i<10; i++)
{
    if (i<5) continue;
    MultiSlider.x[i] = i/9;
}
```

- **while**

A **while** loop allows code to be executed repeatedly based on a given boolean condition. The while structure consists of a condition and a block of code. The condition is first evaluated, and if the condition is true, the block of code within braces is then executed. This repeats until the condition becomes false.

Let's go through another example. In the script below, the **getfirst(Container)** function is used to select the first object within the Container object's hierarchy. Since there're several objects in the Container, the boolean condition (**obj**) will necessarily be true, and therefore the statements within braces executed a first time. The **getobjectrect(obj)** function will extract the position and dimensions of the Container's first object, and its vertical position (**tmp[1]**) recalculated on the basis of the Fader's position, and object's offset (the **offset** variable defines the vertical position within the Container, and is manually entered for each object). Once the **setattrive(obj,'rect',tmp)** function has changed the object's vertical position, the **obj** variable is replaced with the next object down the Container's hierarchy (this is done with the **getnext(obj)** function), and the statements within braces repeated. Those statements will be repeated for each object within the Container, that is, as long as the **getnext(obj)** returns a non-zero value and keep the boolean condition (**obj**) true.

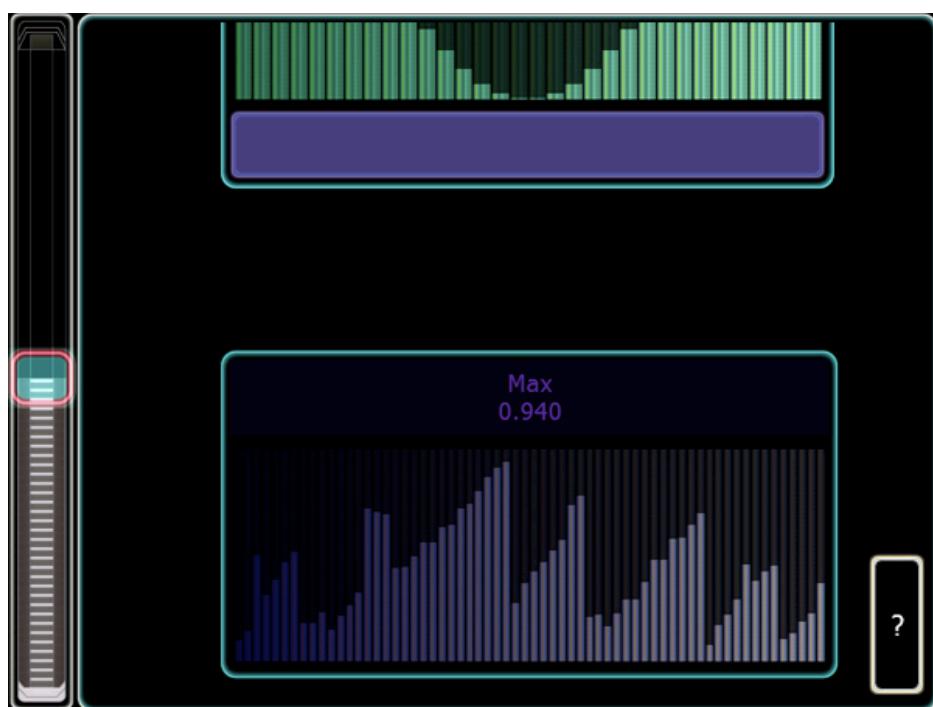


The screenshot shows a software interface with a "SCRIPT" tab at the top. Below it is a code editor window titled "movethem()". The code is as follows:

```

movethem()
Execution : On Expression pos any
decl obj, tmp;
obj=getfirst(Container);
while(obj)
{
    tmp=getobjectrect(obj);
    tmp[1]=pos+obj.offsetY;
    setattrive(obj, 'rect', tmp);
    obj=getnext(obj);
}

```



- **do while**

The **do while** structure also consists of a condition and a block of code, but the difference here is that the block of code within braces is executed first, and then the condition evaluated. If the condition is true, the block of code within braces is executed again. This repeats until the condition becomes false. The do while loop can be referred to as a post-test loop, and the while loop as a pre-test loop.

```
action()
Execution : On Expression x any
decl i=0;
do
{
    MultiSlider.x[i]=i/9;
    i++;
}
while(i<10);
```

## 11.3 Return

The **return** statement isn't specific to conditional or loop structures, but can be used anywhere to instruct the execution to leave the current script. For instance, it can be used to optimize a script's efficiency by preventing the execution to read the entire script needlessly:

```
receive(MIDI_ARGS)
Execution : On MIDI F0 - System Midi 0 0
if (MIDI_ARGS[0]==1)
{
    receiveClipData(MIDI_ARGS);
    return;
}

if (MIDI_ARGS[0]==10)
{
    receiveTrackData(MIDI_ARGS);
    return;
}

if (MIDI_ARGS[0]==3)
{
    receiveClipTriggered(MIDI_ARGS);
    return;
}
```

Now let's go through a last example. The script below can be used to return the maximum value reached by an array (vector). Notice that the script's execution mode is set to **Manual**, so it can be recalled at any time, in any parts of the Project. For instance, it can be recalled in a Monitor Object to display the maximum value reached by a MultiSlider's sliders. This is done by changing the script's arguments (array) by (MultiSlider.x).

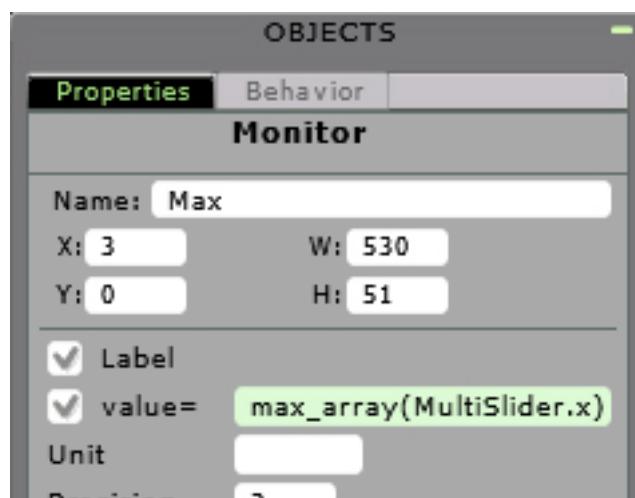
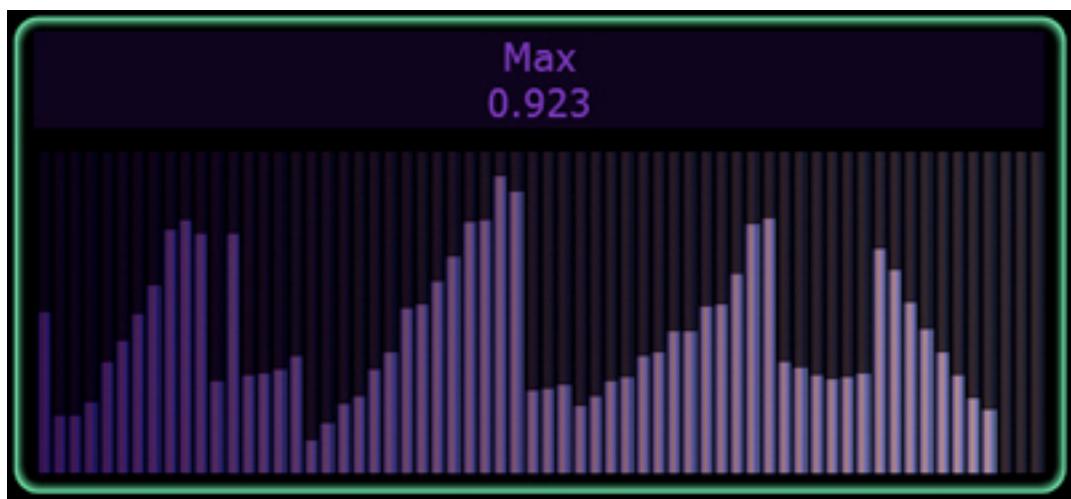
SCRIPT

```
max_array(array)
Execution : Manual

decl size = sizeof(array);
decl i, value;
value = array[0];

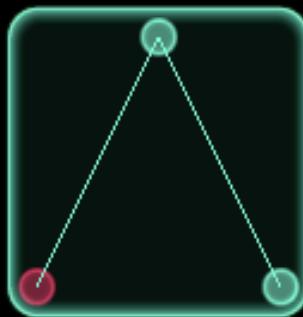
For (i=1; i<size; i++)
{
    if (array[i]>value) value = array[i];
}

return value;
```



# Chapter 12 - Object Reference

## 12.1 Breakpoint



The Breakpoint Object is a multi-segment envelope editor, which assigns each finger to track one of a number of points (up to 64) in a rectangular space. If the edit Option is activated, new points can be created in real-time by double-touching the object's surface. By default the object constraints a point's horizontal displacement so that it does not travel further than its left neighbour ( $x[2] \geq x[1] \geq x[0]$ ), but that can be deactivated by ticking the free form Option.

<b>Variables</b>	
x	A vector containing the points' horizontal positions.
y	A vector containing the points' vertical positions.
<b>Properties</b>	
Name	Name of the Object, also used as its default OSC address.
Label	If checked, the Object's name is displayed on the Object.
Live Edition	If checked, new points can be added by double-touching the object's surface.
Points	Number of points (1 to 64).
Coordinates	If checked, x and y coordinates are displayed when touched.
Background	Choose the color of all the points except the first, together with the outline around the Breakpoint.
First point	Choose the first point's color with the color picker.
Light	Can be a constant, a vector or any mathematical expression and controls the luminosity of the points. -2 means black, +2 means white, and you can choose any decimal number in-between.
<b>Behavior</b>	
Grid	If checked, the range of values produced by the Breakpoint is quantized into [grid] steps. The maximum number of steps for the

		Breakpoint's axis is 33.
Free Form		By default the object constraints a point's horizontal displacement so that it does not travel further than its left neighbour ( $x[2] \geq x[1] \geq x[0]$ ). If the Free Form option is checked, points can be displaced freely on the horizontal axis.
Physics		
• None	Points follow finger positions immediately. Attraction and Friction parameters are ignored.	
• Interpolate	Points follow the fingers' positions according to the value of Attraction. Larger values for attraction (up to 1) cause a point to move to a finger position more quickly. As the Attraction value is lowered, the points take longer to arrive at the finger position. When Attraction is set to 0, points cannot be moved by your fingers.	
• Mass-Spring	Attraction and Friction are both active. Attraction works as described above under Interpolation. Friction ranges between 0 and 1. Lower values of friction mean that if a point is moving it will tend to keep moving. With a value of 0, the points will essentially never stop moving. At a value of 1, a point will move only where you place with your finger. Values of 1 for Attraction and Friction are essentially the same as if Physics is set to Interpolate.	
• Super-Spring	The segments between points also behave as springs, whose initial and resting lengths are set by the Rest parameter.	
Attraction	The amount of attraction the cursors (your fingers) have on the points.	
Friction	The amount of friction the Object's surface applies on the points.	
Speed	This value multiplies the points' speed after Physics computation by a user-defined expression. Input a singleton for the same effect on all points, otherwise input a vector.	
Rest	When Super-Spring mode is enabled, the rest (0 to 1) is the value of the initial and resting lengths of the segments between the points.	
HoldX	If 0, this has no effect. Any value greater than 0 freezes the respective point on its current position on the x-axis. The y-axis remains active. Use a vector, if you want to affect specific points.	
HoldY	If 0, this has no effect. Any value greater than 0 freezes the respective point on its current position on the y-axis. The x-axis remains active. Use a vector, if you want to affect specific points.	
<b>Attributes</b>		
color	{integer*,integer*}	Color of the background and first point
coordinates	0 or 1	Coordinates off/on

editable	0 or 1	live edition off/on
free	0 or 1	free form off/on
grid	0 or 1	grid off/on
grid_steps	{1 to 33, 1 to 33}	number of grid steps on the x and y axis
label	0 or 1	label off/on
nbr	1 to 64	number of points
physic	0 to 3	physics mode
rect	{X,Y,W,H}	object's position and dimensions
zoom	1 to 50	zoom in object

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.2 Container



Containers can enclose any number of objects inside them, including other Containers. A Container can also be made tabbed by ctrl-clicking/right-clicking on it and choosing the option *Make tabbed*. Additional tabs can then be adjoined with the option *Add tab*, and their respective content displayed by touching the corresponding tab. The tabs' order can be modified by ctrl-clicking/right-clicking within a Container and choosing *Bring up* or *Bring down*.

Properties		
Name	The Container's name (used as a prefix for addressing the Objects inside the Container, as in <i>Container/Fader.x</i> ).	
Lock	A locked Container prevents editing and display of its content in the Project panel.	
Transparent	If checked, the Objects lying "under" the Container (but not belonging to its content) are shown and the Container's frame is not displayed anymore.	
Color	The color of the Container's frame.	
Attributes		
color	integer*	color of the container
label	0 or 1	label off/on
rect	{X,Y,W,H}	object's position and dimensions
transparent	0 or 1	transparent off/on

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.3 Custom Button



The Custom Button can act as a pad or switch. You can set the text of the button for on and off state. Another option is to have the state displayed by two different geometric forms to be chosen among.

### Variables

x	The state of the button.
---	--------------------------

### Properties

Name	The name of the Object, also used as its default OSC address.
Style Off	A menu for choosing how the Off state is depicted. Choose between text or symbols. If text is chosen, a whitespace can be used in the text entry.
Style On	A menu for choosing how the On state is depicted.
Font	A menu for choosing the font size for the text. The font size ranges from 8pt to 24pt.
Alignment	A graphical menu for choosing the position of the text within the boundaries of the Object. You get a choice between 9 different positions. This feature does not work for the symbols.
Color Off/On	Dial in the color for the two respective button states.
Light	Can be a constant or any mathematical expression and controls the luminosity of your button. -2 means black, +2 means white, and you get to choose any decimal number in-between.

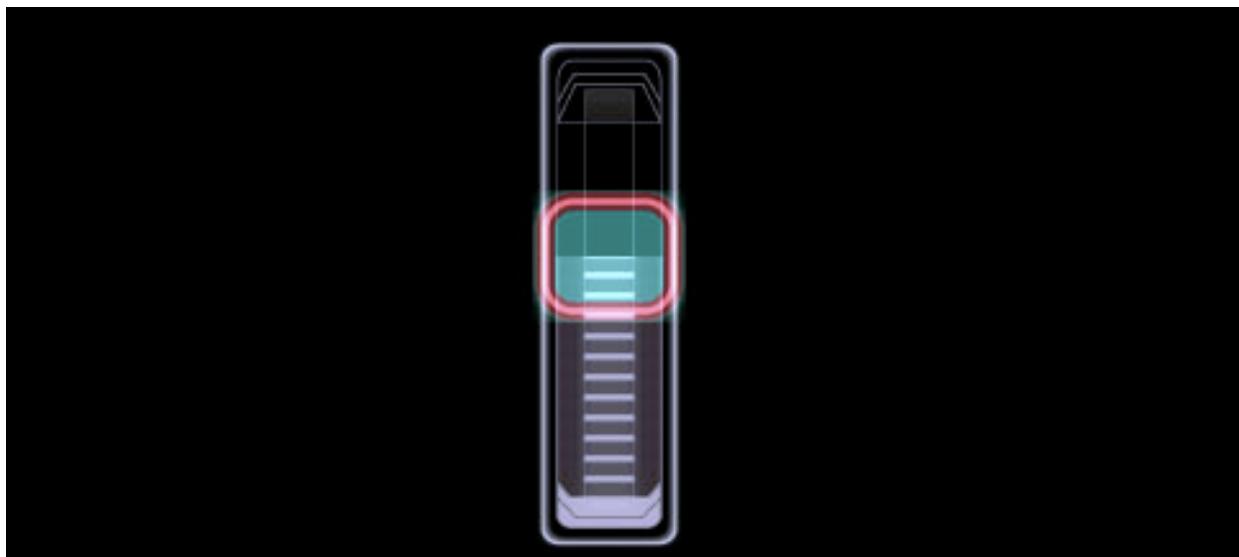
### Behavior

Capture	If Capture is checked, an Object will only react to cursors created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is off, the original behavior is restored, meaning an Object will react to whatever cursor is present at any moment in its area.
Mode	A menu for choosing if the Button works as a Switch or a Pad. A Switch changes state on touch and doesn't change back on removal of the finger. A Pad changes its state back if you remove your finger.

<b>Attributes</b>		
behavior	0 or 1	switch or pad mode
bitmap	{0 to 14,0 to 14}	decoration bitmap in off state, in on state
capture	0 or 1	capture off/on
color	{integer*,integer*}	color off state, color on state
label_off	text	button text in off state
label_on	text	button text in on state
outline	0 or 1	outline off/on
rect	{X,Y,W,H}	object's position and dimensions

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.4 Fader



The Fader tracks your finger with a virtual cap and generates a value corresponding to the position of the cap on the fader. The Fader can be oriented vertically or horizontally. Grab and drag a corner to change its orientation.

### Variables

x	The location of the cap. When the cap is at the top or right-most position of the fader (depending on orientation), the value is 1 by default. When it is at the bottom or at the left-most position, respectively, the value is 0.
z	A flag variable for detecting if the Fader is being touched. 1 if touched, 0 if untouched. This is useful to emulate fader touch of control surfaces.

### Properties

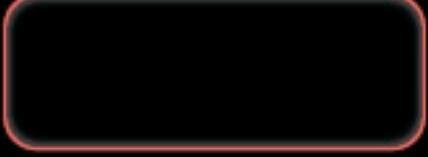
Name	The name of the Breakpoint Object, also used as its default OSC address.
Label	If checked, the Object's name is displayed on the Interface.
Value	If checked, the current value of the Fader is displayed on the Interface. In addition, you can enter a formula for how the value is displayed. This does not affect the actual value sent by the Fader, which remains between 0 and 1.
Unit	This user-specified text is appended at the end of the value display. Use it to specify the type of value, as in dB or ms.
Precision	Specifies the number of decimal places for the value display. The default value is 3 and the maximum number is 6. This setting has no influence on the actual output of the Object. You have to scale the output using expressions or on the Target side.
Color	Pick a color for the fader's body, its cap always keeps the pink outline.

<b>Behavior</b>	
Grid	If checked, the range of values produced by the Fader is quantized into [grid] steps. The maximum number of steps for the Fader is 33.
Capture	If Capture is checked, an Object will only react to cursors that were created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is off, the old school way from previous versions is restored, meaning an Object will react to whatever cursor is present at any moment in its area.
Cursor Mode	<ul style="list-style-type: none"> <li>• Limited      The Fader will respond to a new cursor only if the original one has been destroyed (i.e. finger is raised)</li> <li>• Get Newer    Whenever a new cursor appears inside the Object's area, it gains full control of the Object</li> <li>• Barycentric    Each cursor, old or new, has the same amount of influence on the Object.</li> <li>• Cap Only     The Object acts like a conventional fader that doesn't react to cursors outside of the cap area</li> </ul>
Physics	<ul style="list-style-type: none"> <li>• None        The Fader cap tracks one finger immediately. If other fingers touch the fader, they are ignored.</li> <li>• Interpolate    The cap moves according to the value of Attraction from its current location to the location of your finger. Larger values for attraction (up to 1) cause the cap to move to the new finger location more quickly. When Attraction is set to 0, the cap cannot be moved by your finger.</li> <li>• Mass-Spring    Attraction and Friction are both active. Friction ranges between 0 and 1. Lower values of friction mean that if the cap is moving it will tend to keep moving. With a value of 0, the cap will never stop moving. At a value of 1, the cap exactly follows your finger. Values of 1 for Attraction and Friction are essentially the same as if Physics is set to None.</li> </ul> <p>Please consider that the Lemur will send the Fader's position constantly with lower values of Attraction and Friction in Mass-Spring mode.</p>
Attraction	The amount of attraction the cursor (your finger) has on the Fader
Friction	See Mass-Spring above
<b>Attributes</b>	

capture	0 or 1	capture off/on
color	integer*	color of the object
cursor	0 to 3	cursor mode
grid	0 or 1	grid off/on
grid_steps	1 to 33	number of grid steps
label	0 or 1	label off/on
physic	0 to 2	physics mode
precision	0 to 6	decimal places of displayed value
rect	{X,Y,W,H}	object's position and dimensions
unit	text	unit of displayed value
value	0 or 1	value off/on
zoom	1 to 50	zoom in object

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.5 Gesture



The Gesture Object is a trackpad control with advanced gesture recognition such as pinch, rotation and traces. Its x and y variables react like those of a traditional trackpad, taking acceleration into account, and can be routed to a Target's KbMouse port to take over your computer's mouse pointer.	
<b>Variables</b>	
angle	A vector containing three values: the angle in radian between two fingers (one finger in the centre of the Object, the other rotating around it), followed by the x and y coordinates of the central finger.
pan	A vector containing four values: how much a single finger has moved along the horizontal and vertical axis during a frame, initial x and y positions (y position value is inversed).
pinch	A value above 1.0 for two fingers pinching towards the exterior, and below 1.0 for two fingers pinching towards the centre.
x	A value corresponding to the relative horizontal motion and position of a finger on the object's surface. This value can be routed to a Target's KbMouse port and take over your computer's mouse pointer through the JazzDaemon. Its minimal and maximal values can be constrained to fit a given screen resolution. This value is also proportional to the Acceleration coefficient.
y	A value corresponding to the relative vertical motion and position of a finger on the object's surface. This value can be routed to a Target's KbMouse port and take over your computer's mouse pointer through the JazzDaemon. Its minimal and maximal values can be constrained to fit a given screen resolution. This value is also proportional to the Acceleration coefficient.
<b>Properties</b>	
Name	The name of the Gesture Object, also used as its default OSC address.
Transparent	If checked, the Object's frame is no longer displayed.
Color	Pick a color for the Object's frame.

<b>Behavior</b>		
Constrained	If checked, constrains output inside chosen values.	
Acceleration	A coefficient between 0 and 1 scaling the x and y variables.	
Per-axis speed	If checked, adapts the speed to the Object's dimensions on a "per-axis" basis.	
<b>Attributes</b>		
acceleration	0 to 1	acceleration coefficient of x and y variables
boundaries	{0 to ,0 to ,0 to ,0 to }	boundaries of x and y variables
color	integer*	color of the object
constrained	0 or 1	constrained off/on
rect	{X,Y,W,H}	object's position and dimensions
speed	0 to 2	cursor speed
transparent	0 or 1	off/on

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.6 Knob



The Knob Object emulates two types of knobs: Classic knobs constrained between a min and max value (0 and 1 by default), and endless encoders that can reach any value through multiple successive turns.

### Variables

x	The current value of the knob, based on absolute angle position in Classic mode, or rotation count in Endless mode.
---	---

### Properties

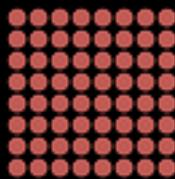
Name	The name of the Knob Object, also used as its default OSC address.
Endless Knob	If checked, the knob switches to endless mode.
Label	If checked, the Object's name is displayed above the knob.
Value	If checked, the current value of the Knob is displayed on the Interface. In addition, you can enter a formula for how the value is displayed. This does not affect the actual value sent by the Knob, which remains between 0 and 1.
Unit	This user-specified text is appended at the end of the value display. Use it to specify the type of value, as in dB or ms.
Precision	Specifies the number of decimal places for the value display. The default value is 3 and the maximum number is 6. This setting has no influence on the actual output of the Object. You have to scale the output using expressions or on the Target side.
Background	Pick the Knob's background color.
Foreground	Pick the Knob's foreground color.

Behavior	
Grid	If checked, the range of values produced by the Knob is quantized into [grid] steps. The maximum number of steps for the Fader is 33.
Capture	If Capture is checked, an Object will only react to cursors that were created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is off, the old school way from previous versions is restored, meaning an Object will react to whatever cursor is present at any moment in its area.
Cursor Mode	<ul style="list-style-type: none"> <li>• Limited                          The Fader will respond to a new cursor only if the original one has been destroyed (i.e. finger is raised)</li> <li>• Get Newer                        Whenever a new cursor appears inside the Object's area, it gains full control of the Object</li> <li>• Barycentric                      Each cursor, old or new, has the same amount of influence on the Object.</li> <li>• Cap Only                        The Object acts like a conventional fader that doesn't react to cursors outside of the cap area</li> </ul>
Physics	<ul style="list-style-type: none"> <li>• None                             The Fader cap tracks one finger immediately. If other fingers touch the fader, they are ignored.</li> <li>• Interpolate                     The cap moves according to the value of Attraction from its current location to the location of your finger. Larger values for attraction (up to 1) cause the cap to move to the new finger location more quickly. When Attraction is set to 0, the cap cannot be moved by your finger.</li> <li>• Mass-Spring                    Attraction and Friction are both active. Friction ranges between 0 and 1. Lower values of friction mean that if the cap is moving it will tend to keep moving. With a value of 0, the cap will never stop moving. At a value of 1, the cap exactly follows your finger. Values of 1 for Attraction and Friction are essentially the same as if Physics is set to None.  Please consider that the Lemur will send the Fader's position constantly with lower values of Attraction and Friction in Mass-Spring mode.</li> </ul>
Attraction	The amount of attraction the cursor (your finger) has on the Fader
Friction	See Mass-Spring above

<b>Attributes</b>		
color	{integer*,integer*}	color of background, color of foreground
cursor	0 to 2	cursor mode – Limited, Barycentric, Get Newer
grid	0 or 1	grid off/on
grid_steps	1 to 33	number of grid steps
label	0 or 1	label off/on
mode	0 or 1	polar or linear
physic	0 to 2	physics mode – None, Interpolate, Mass-Spring
precision	0 to 6	decimal places of displayed value
rect	{X,Y,W,H}	object's position and dimensions
unit	text	unit of displayed value
value	0 or 1	value off/on

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.7 Leds



The Leds Object is a two-dimensional array of Leds. It can be a single led or a whole matrix of them, all with their individual light and value parameters. Remember that those properties can be set to vectors when dealing with several columns or rows of Leds.

The Leds are transparent to touch but not transparent in terms of display. This means you can implement switch or pad functionality by placing a hidden Switches or Pads Object underneath the Leds.

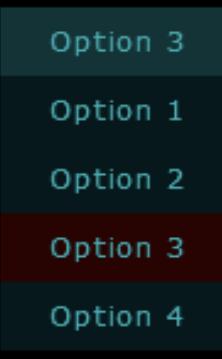
### Properties

Name	The name of the Leds Object, also used as its default OSC address.
Label	If checked, the Object's name is displayed above the Leds.
Transparent	If checked, the Leds'background is transparent.
Value	In Bargraph mode this value represents the percentage of the Leds being switched on. If Bargraph mode is off, only 1 and 0 are accepted as values. If you send a single value (0,1) it will switch all Leds to their on or off colors. If you have multiple rows/columns, you can use a vector to individually address the Leds.
Columns	The number of columns of Leds contained in the Object. Only 16 columns of Leds can be set.
Rows	The number of rows of Leds contained in the Object. Only 16 rows of Leds can be set.
Multicolor	If checked, the 'colors' attribute of the Object can be set to a different value for each Led.
BarGraph	The Bar Graph mode makes a matrix or vector of Leds act as your typical bar graph: the Leds work together to graphically display the current value.
Color Off	Pick a color for the Leds' off-state.
Color On	Pick a color for the Leds' on-state.
Light	Can be a constant, a vector or any mathematical expression and controls the luminosity of your Objects. -2 means black, +2 means white, and you can choose any decimal number in-between.

<b>Attributes</b>		
bagraph	0 or 1	bagraph off/on
color	{integer*,integer*}	color off, color on
column	0 to 16	number of columns
label	0 or 1	label off/on
rect	{X,Y,W,H}	object's position and dimensions
row	0 to 16	number of rows
transparent	0 or 1	transparent off/on

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.8 LemurMenu

	The LemurMenu Object is a “pop-up” menu selection tool. It can hide and display a list of up to 32 individual text items. By default the object outputs the index of the selection.	
<b>Variables</b>		
selection	The current index of the selection (1,2,3,4...)	
<b>Properties</b>		
Name	The name of the LemurMenu Object, also used as its default OSC address.	
Scale Output	Scale Output to Midi values.	
Transparent	If checked, the Object’s background becomes transparent.	
Font	Text content font size.	
Alignment	A graphical menu for choosing the position of the text within the boundaries of the LemurMenu’s cells.	
Color	Color of the the Object	
Items	A list for editing the menu’s content.	
<b>Attributes</b>		
color	integer*	object’s color
content	{text,text...}	menu items
rect	{X,Y,W,H}	object’s position and dimensions
scale	0 or 1	scale output off/on
transparent	0 or 1	transparent off/on

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.9 Monitor

	
The Monitor produces no data when you touch it. Its purpose is to display information sent to the Lemur by your computer.	
<b>Variables</b>	
selection	The current index of the selection (1,2,3,4...)
<b>Properties</b>	
Name	The name of the LemurMenu Object, also used as its default OSC address.
Label	If checked, the Object's name is displayed above the value.
Value	The checkbox is not functional. The text field next to Value represents the monitor's default value. Since any value can be sent to the monitor, there is no 0-1 limitation.
Unit	Appends arbitrary text to the end of the value display.
Precision	Number of floating-point digits that appear. A precision of 0 displays only the integer part of numbers the monitor receives.
Font	A menu for choosing the font size for the label and value of the Monitor. The font size ranges from 8pt to 24pt.
Alignment	A graphical menu for choosing the position of the text within the boundaries of the Object. You get a choice between 9 different positions.
Transparent	If checked, only the label and the value is displayed and the Monitor's background becomes transparent. This also makes it possible to use the Monitor as a text label anywhere on the Interface.
Color	Pick the Object's color.

<b>Attributes</b>		
color	integer*	color of the object
label	0 or 1	label off/on
precision	0 to 6	decimal places of displayed value
rect	{X,Y,W,H}	object's position and dimensions
transparent	0 or 1	transparent off/on
unit	text	unit of displayed value
value	0 or 1	value off/on

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.10 MultiBall



The MultiBall Object assigns each finger to track one of a number of balls (up to 10) in a rectangular space. Balls can either always be visible or only appear when you touch the space; the latter is called ephemeral mode. The brightness of the balls is sent as the z variable in the Object.

### Variables

x	A list of the horizontal positions of all the balls.
y	A list of the vertical positions of all the balls.
z	A list of the brightness values of all the balls. Brightness values change only when the MultiBall Object is in ephemeral mode.

### Properties

Name	The name of the MultiBall Object, also used as its default OSC address.
Label	If checked, the Object's name is displayed on the Interface.
Numbers	If checked, a running number is displayed on each ball.
Balls	Number of balls (1 to 10)
Color	Pick a color for the MultiBall's frame,

### Behavior

Grid	If checked, the range of values produced by the Balls is quantized into [grid] steps. The maximum number of steps for the MultiBall axis is 33.
Ephemeral	If checked, the MultiBall behaves in a mode where the balls disappear until you touch it with one or more fingers. The brightness of the balls becomes the value of the z variable of the Object, and the way the brightness changes over time is controlled by an ADSR envelope (described below). When Ephemeral is not checked, the balls are always visible and their z

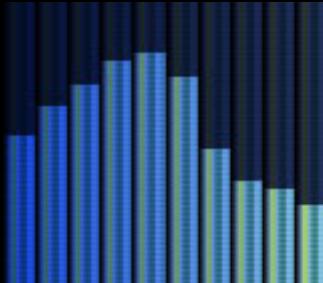
	values are constantly 1.
Capture	If Capture is checked, an Object will only react to cursors that were created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is off, an Object will react to whatever cursor is present at any moment in its area.
Cursor Mode	<ul style="list-style-type: none"> <li>• Limited    The Balls will respond to a new cursor only if the original one has been destroyed (i.e. finger is raised)</li> <li>• Barycentric    Each cursor has the same amount of influence on the Object.</li> <li>• Get Oldest Ball                                      If all the balls are currently “possessed” by your fingers and a new cursor comes in, it will take control of the oldest ball (the one that lived the longest in Ephemeral, or the first ball in normal mode).</li> <li>• Get Closest Ball                                      If all the balls are currently “possessed” by your fingers and a new cursor comes in, it will take control of the closest ball.</li> </ul>
Physics	<ul style="list-style-type: none"> <li>• None    Balls move to finger positions immediately, and the settings of Attraction and Friction are ignored.</li> <li>• Interpolate    Balls move toward finger positions according to the value of Attraction. Larger values for attraction (up to 1) cause a ball to move to a finger position more quickly. As the Attraction value is lowered, the balls take longer to arrive at the finger position. When Attraction is set to 0, your fingers can't move balls.</li> <li>• Mass-Spring    Attraction and Friction are both active. Attraction works as described above under Interpolation. Friction ranges between 0 and 1. Lower values of friction mean that if a ball is moving it will tend to keep moving. With a value of 0, the balls will essentially never stop moving. At a value of 1, a ball will move only where you touch with your finger. Values of 1 for Attraction and Friction are essentially the same as if Physics is set to Interpolate.</li> </ul>
Attraction	Amount of attraction the cursor (your fingers) have on the Balls.
Friction	See Mass-Spring above
Speed	This value multiplies the balls' speed after Physics computation by a user-defined expression. Input a singleton for the same effect on all balls, otherwise input a vector. Experiment with negative values for crazy effects.
Attack	Applies only when using ephemeral mode. The Attack value specifies the number of seconds over which the z variable (brightness of a ball) increases from its initial value of 0 to a maximum of 1 after you touch the screen. As an example, if the Attack value is 0, the ball will be at full brightness the moment

		you touch the screen.
Decay		Applies only when using ephemeral mode. The Decay value specifies the number of seconds over which the brightness will decrease after the initial Attack portion of the envelope has completed. During the Decay portion of the envelope, the z variable (brightness of a ball) will decrease from 1 to the level set by the Sustain value.
Sustain		Applies only when using ephemeral mode. The sustain value is the level (between 0 and 1) at which the brightness of the ball will remain as long the MultiBall Object is tracking your finger within its space. The Sustain level is reached after the Attack and Decay portion of the envelope have completed. If your finger lifts up from the touch surface before the completion of the Attack and/or Decay portion of the envelope, the Release portion of the envelope is triggered immediately after the Decay portion completes, and the brightness ultimately goes to 0.
Release		Applies only when using ephemeral mode. The Release value specifies the number of seconds over which the brightness of a ball will decrease from its Sustain level to 0, starting at the moment that you lift up ("release") your finger from the touch surface.
Hold		Its effect is similar to a sustain pedal, freezing the Object's state as long as its value is 1. When set to 0, it has no effect. This means this parameter should be used with a mathematical expression depending on other Objects. For instance, if you have a Switch Object named Sustain in your interface, you can set the hold parameter of a Pad to Sustain.x so the Switch gets the ability to freeze the current lightness.
HoldX		If 0, this has no effect. Any value greater than 0 freezes the respective ball on its current position on the x-axis. The y-axis remains active. Use a vector, if you want to affect specific balls.
HoldY		If 0, this has no effect. Any value greater than 0 freezes the respective ball on its current position on the y-axis. The x-axis remains active. Use a vector, if you want to affect specific balls.
<b>Attributes</b>		
capture	0 or 1	capture off/on
color	integer*	color of the object
cursor	0 or 1	cursor off/on
ephemere	0 or 1	ephemeral off/on
grid	0 or 1	grid off/on
grid_steps	{1 to 33, 1 to 33}	number of grid steps on the x and y axis

label	0 or 1	label off/on
nbr	1 to 10	number of balls
physic	0 to 2	physics mode – None, Interpolate, Mass-Spring
polyphony	0 or 1	polyphony off/on
rect	{X,Y,W,H}	object's position and dimensions
zoom	1 to 50	zoom in object

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.11 MultiSlider



The MultiSlider Object tracks movement across an array of sliders. You can “wipe” all the faders to a set value with one horizontal gesture. This is pretty hard to do with real—or virtual—faders.

### Variables

x	A list of the vertical positions of all the individual sliders.
---	---

### Properties

Name	The name of the Object, also used as its default OSC address.
Horizontal	Swaps the MultiSlider’s orientation from vertical to horizontal.
Bipolar	If checked, the Object’s zero position is the slider’s mid-length.
Label	If checked, the Object’s name is displayed on the Interface.
Slider	Number of sliders (1 to 64)
Gradient	If checked, a gradient is applied on the Object’s color.
Multicolor	If checked, the ‘colors’ attribute is active and can be used to change the colors of individual sliders.
Color	Drag the color bar to change the “thematic” color of the sliders.
Light	Can be a constant, a vector or any mathematical expression and controls the luminosity of your Objects. -2 means black, +2 means white, and you get to choose any decimal number in-between.

### Behavior

Grid	If checked, the range of values produced by the Sliders is quantized into [grid] steps. The maximum number of steps for the Fader is 33.
Capture	If Capture is checked, an Object will only react to cursors that were created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is

	off, the old school way from previous versions is restored, meaning an Object will react to whatever cursor is present at any moment in its area.	
Physic	If checked, the MultiSlider emulates the physics of an Object similar to a plucked string anchored at the left and right sides of the array of sliders. Your fingers “pluck” the string by lifting it up in one or more places. The values of the sliders ramp up to meet your fingers and track them as they move. Lifting your finger(s) from the surface releases the string, and its subsequent behavior is determined by the Tension, Friction, and Height values.	
Tension	A value between 0 and 1 corresponding to the tension on a string. As tension increases, the frequency of oscillation of the string increases. Increasing the tension is something like turning the tuning peg of a guitar to raise the pitch of a string.	
Friction	A value between 0 and 1 corresponding to the damping on a string. As friction increases, the damping on the oscillation increases. With large friction values, the string returns to its resting position quickly. With smaller friction values, the string may oscillate for a long time.	
Height	When Tension is enabled, the height (0 to 1) is the value of the initial and resting position of the string.	
<b>Attributes</b>		
capture	0 or 1	capture off/on
color	integer*	color of the object
grid	0 or 1	grid off/on
grid_steps	1 to 33	number of grid steps
horizontal	0 or 1	horizontal off/on
label	0 or 1	label off/on
multicolor	{integer*,integer*,...}	color of individual sliders
nbr	1 to 64	number of sliders
physic	0 or 1	physics off/on
rect	{X,Y,W,H}	object's position and dimensions

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.12 Pads



The Pads Object is a two-dimensional array of buttons that are triggered by touch. They are intended to trigger events instead of represent state, since they eventually return to an "off" value after you touch them.

### Variables

x	A list of the envelope (brightness) values of the pads.
---	---

### Properties

Name	Name of the Pads Object, also used as its default OSC address.
Label	If checked, the Object's name is displayed on the Interface.
Numbers	If checked, each pad is labelled with a number corresponding to the order in which its value is transmitted, starting with 0.
Columns	The number of columns of pads contained in the Object (max 16).
Rows	The number of rows of pads contained in the Object (max 16).
Multilabel	If checked, the 'labels' attribute is active, and each pad can be labelled individually.
Multicolor	If checked, the Object's 'colors' attribute is active, and each Pad can take its own color value.
Color Off	Color for the Object's off state.
Color On	Color for the Object's on state.
light	Can be a constant, a vector or any mathematical expression and controls the luminosity of your Objects. -2 means black, +2 means white, and you get to choose any decimal number in-between.

### Behavior

Capture	If Capture is checked, an Object will only react to cursors that were created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is off, an Object will react to whatever cursor is present at any moment in its area.
---------	--

Attack	The Attack value specifies the number of seconds over which the x variable (pad brightness) increases from its initial value of 0 to a maximum of 1 after you touch the screen. As an example, if the Attack value is 0, the pad will be at full brightness the moment you touch the screen. An attack value of 10 means the pad will take 10 seconds to reach the full value.
Decay	The Decay value specifies the number of seconds over which the x variable (pad brightness) will decrease after the initial Attack portion of the envelope has completed. During the Decay portion of the envelope, the x variable (pad brightness) will decrease from 1 to the level set by the Sustain value.
Sustain	The Sustain value is the level (between 0 and 1) at which the x variable (pad brightness) will remain as long your finger is touching the pad. The Sustain level is reached after the Attack and Decay portion of the envelope have completed. If your finger lifts up from the touch surface before the completion of the Attack and/or Decay portion of the envelope, the Release portion of the envelope is triggered immediately after the Decay portion completes, and the brightness ultimately goes to 0.
Release	The Release value specifies the number of seconds over which the x variable (pad brightness) will decrease from its Sustain level to 0, starting at the moment that you lift up ("release") your finger from the touch surface.
Hold	Its effect is similar to a sustain pedal, freezing the Object's state as long as its value is 1. When set to 0, it has no effect. This means this parameter should be used with a mathematical expression depending on other Objects. For instance, if you have a Switch Object named Sustain in your interface, you can set the hold parameter of a Pad to Sustain.x so the Switch gets the ability to freeze the current lightness.

### Attributes

capture	0 or 1	capture off/on
color	{integer*,integer*}	color off state,color on state
colors	{integer*,integer*}	Off state color values for each Pads
column	1 to 16	number of columns
label	0 or 1	grid off/on
labels	{Text,Text,..}	label
multicolor	0 or 1	Multicolour off/on

multilabel	0 or 1	Multilabel off/on
rect	{X,Y,W,H}	object's position and dimensions
row	1 to 16	number of rows

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.13 Range



The Range Object is a slider with adjustable length. Touch either end to change the width/height of the range.

### Variables

x	A vector containing the lower and higher positions.
---	---

### Properties

Name	The name of the Range Object, also used as its default OSC address.
Horizontal	Swaps the orientation of the Range from vertical to horizontal.
Label	If checked, the Object's name is displayed on the Interface.
Color	Pick the color of the Object
Light	Can be a constant, a vector or any mathematical expression and controls the luminosity of your Objects. -2 means black, +2 means white, and you get to choose any decimal number in-between.

### Behavior

Grid	If checked, the range of values produced by the Sliders is quantized into [grid] steps. The maximum number of steps for the Fader is 33.
Capture	If Capture is checked, the Object will only react to cursors that were created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is off the Object will react to whatever cursor is present at any moment in its area.
Physics	If checked, the Range boundaries snap back to user-defined positions (that can evolve based on other variables) when you release them. Physics behavior is determined by the Tension, Friction, and Min_Height and Max_Height values.
Tension	A value between 0 and 1 corresponding to the tension of the string that links the boundaries to their snap-back positions. You can enter a vector to specify different tensions for the two

		extremities.
Friction	A value between 0 and 1 corresponding to the damping on a string. As friction increases, the damping on the oscillation increases. With large friction values, the string returns to its resting position quickly. With smaller friction values, the string may oscillate for a long time. You can enter a vector to specify different frictions for the two extremities.	
Min_height	When Physics mode is enabled, the height (0 to 1) is the value of the initial and resting position of the minimum boundary of the Range.	
Max_height	When Physics mode is enabled, the height (0 to 1) is the value of the initial and resting position of the maximum boundary of the Range.	
Drag	If drag is not zero, the current range can be dragged with the finger without changing its limits. You may use this with a switch to toggle the behaviour of your Range Object (put Switch.x into the drag field). The field can contain any expression.	
<b>Attributes</b>		
capture	0 or 1	capture off/on
color	integer*	color of the object
grid	0 or 1	grid off/on
grid_steps	1 to 33	number of grid steps
horizontal	0 or 1	horizontal off/on
label	0 or 1	label off/on
multicolor	{integer*,integer*,...}	Color of individual sliders
nbr	1 to 64	number of sliders
physic	0 or 1	physics off/on
rect	{X,Y,W,H}	object's position and dimensions

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.14 RingArea



The RingArea tracks your finger inside a circular space. It reports the X and Y coordinates of a ball that can be programmed to have a variable degree of attraction toward a central point. You can specify the location of the attraction point within the circular space.

### Variables

x	The horizontal position of the ball.
y	The vertical position of the ball.

### Properties

Name	The name of the RingArea Object, also used as its default OSC address.
------	--

Label	If checked, the Object's name is displayed above the circular space.
-------	--

Color	Pick a color for the RingArea.
-------	--------------------------------

### Behavior

Capture	If Capture is checked, the Object will only react to cursors that were created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is off, the Object will react to whatever cursor is present at any moment in its area.
---------	--

Attraction	A value from 0-1 representing the speed of the ball from its attraction point to your finger when you touch the Object, and from your finger to the attraction point when you release your finger.
------------	--

Friction	A value from 0-1 representing the stickiness of the movement to either your finger or the attraction point. Lower values of friction make the ball overshoot the attraction point when it approaches, causing bouncing and/or oscillation.
----------	--

Attractor X	A value from 0-1 representing the horizontal location of the attraction point within the Object's rectangle. A zero value of Attractor X is at the left edge and a value of 1 is at the right edge.	
Attractor Y	A value from 0-1 representing the vertical location of the attraction point within the Object's rectangle. A zero value of Attractor Y is at the bottom edge and a value of 1 is at the top edge.	
<b>Attributes</b>		
capture	0 or 1	capture off/on
color	integer*	color of the object
label	0 or 1	label off/on
rect	{X,Y,W,H}	object's position and dimensions

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.15 SignalScope



The SignalScope displays values of other Objects and variables on your Lemur. The “trace” shows a recent history of the value of what you are monitoring. It is transparent to touch, meaning that you can place it on top of Objects and still interact with them.

### Properties

Name	The name of the SignalScope Object, also used as its default OSC address.
Label	If checked, the Object’s name is displayed in the Scope area.
Mode XY	If checked, the signal scope shows both an X and Y value plotted against each other. If unchecked, the X value is time, shifting Y values to the left.
Transparent	If checked the Scope’s frame is no longer visible.
Color	Pick a color for the Scope and its frame.

### Behavior

X	If Mode XY is enabled, X can be the value of a variable or a constant; otherwise it is assigned to time against which the Y value will be plotted.
Y	A variable or constant expression that will be periodically evaluated and plotted against X. For example, to plot the x variable of a Fader Object called chan1, enter chan1.x in the text field.
Time base	Corresponds to the time (in seconds) displayed on the scope. As the Time base increases, individual elements of the graph will decrease in width as more of the “past” is shown.

### Attributes

color	integer*	color of the object
label	0 or 1	label off/on
mode	0 or 1	mode XY off/on
rect	{X,Y,W,H}	object's position and dimensions
transparent	0 or 1	transparent off/on

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.16 SurfaceLCD

Track 1   Track 2   Track 3   Track 4   Track 5   Track 6   Track 7   Track 8								
0   0   0   0   0   0   0   0   0								
The SurfaceLCD Object emulates traditional control surfaces' LCD by making use of the MIDI input capabilities of the Lemur to display track names, parameter changes, timecode, all updated in real-time by your Digital Audio Workstation. The SurfaceLCD is compatible with the Mackie Control surface protocol.								
<b>Properties</b>								
Name	The name of the SurfaceLCD Object, also used as its default OSC address.							
Transparent	If checked, the background of the SurfaceLCD disappears to obtain a see-through appearance.							
Target	This lets you select the MIDI Target number for information display. First, setup a MIDI target with an input of your liking inside the JazzEditor settings. Then, configure your DAW to send Mackie Control surface data to this MIDI input. Refer to the bundled examples for mappings of control Objects compatible with control surfaces protocols known by your DAW.							
Display								
Main LCD	In this mode, the Object displays Track Information.							
Timecode	In this mode, the Object displays up-to-date Timecode information.							
Assignment	In this mode, the Object displays a three characters code describing the current assignment of other mapped control Objects.							
<b>Attributes</b>								
color	integer*	color of the object						
label	0 or 1	label off/on						
mode	0 or 1	mode XY off/on						
rect	{X,Y,W,H}	object's position and dimensions						
transparent	0 or 1	transparent off/on						

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.17 Switches



The Switches Object is a two-dimensional array of toggle switches, representing and transmit one or more on-off states.

### Variables

x	A list of the on-off values of the switches in the Object. The list starts with the top-left corner, and traverses the first row before starting at the beginning of the second row.
---	--

### Properties

Name	Name of the Object, also used as its default OSC address.
Label	If checked, the Object's name is displayed on the Interface.
Numbers	If checked, each switch is labelled with a number corresponding to the order in which its value is transmitted, starting with 0.
Radio	In Radio mode, only one switch can be turned on at any particular time. Turning on any switch turns all the others off.
Columns	Number of columns of switches contained in the Object (max 16).
Rows	Number of rows of switches contained in the Object (max 16).
Multicolor	If checked, the Pads 'colors' attribute is active, and color values can be set for each Pad individually.
Color Off	Pick a color for the Object's off-state
Color On	Pick a color for the Object's on-state
light	Can be a constant, a vector or any mathematical expression and controls the luminosity of your Objects. -2 means black, +2 means white, and you get to choose any decimal number in-between.

### Behavior

Capture	If Capture is checked, an Object will only react to cursors that were created inside its area. Even if the cursor later leaves the Object for another position, it will remain in control of the original Object, until it is destroyed eventually. When Capture is off, an Object will react to whatever cursor is present at any moment in its area.
---------	--

Paint	If this flag is active, you can “paint” on an array of switches by dragging your finger around. If paint is inactive, a touch only toggles the switch you hit first and dragging the finger around has no further effect.	
<b>Attributes</b>		
capture	0 or 1	capture on/off
color	{integer*,integer*}	color off state,color on state
column	1 to 16	number of columns
label	0 or 1	label off/on
paint	0 or 1	paint off/on
radio	0 or 1	radio off/on
rect	{X,Y,W,H}	object's position and dimensions
row	1 to 16	number of rows

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

## 12.18 Text



Lemurize !

The Text Object produces no data when you touch it. Its purpose is to display arbitrary text including whitespace characters typed in via the JazzEditor.

### Properties

Name	The name of the Text Object, also used as its default OSC address.
Transparent	If checked, only the text is displayed and the Object's background becomes transparent.
Text	Type in any text you want the Object to display for you.
Font	A menu for choosing the font size for the displayed text. The font size ranges from 8pt to 24pt.
Alignment	A graphical menu for choosing the position of the text within the boundaries of the Object. You get a choice between 9 different positions.
Color	Pick a color for the Object
light	Can be a constant, a vector or any mathematical expression and controls the luminosity of your Objects. -2 means black, +2 means white, and you get to choose any decimal number in-between.

### Attributes

color	integer*	color of the object
content	text	displayed text
rect	{X,Y,W,H}	object's position and dimensions
transparent	0 or 1	transparent off/on

\*the value of the color attribute can range from 0 to 8355711:  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127

# Chapter 13 - Parser Reference

## 13.1 Vectors and Singletons

The Parser handles two kinds of variables: **vectors** and **singletons**. A vector (also known as an array or list), is a mathematical value that holds more than one element (number), while a singleton is a value containing a single value:

a vector :{0, 2, 0.5}

a singleton :3

The Lemur expression syntax allows operations on vectors and individual access to their elements. If you want to access the items of a vector separately, use brackets [].

a = {1,3,4}

a[0] = 1

a[2] = 4

This also works for multiple items:

a[0,2] = {1, 4}

Typically, the Variables of the Breakpoint, Leds, MultiSlider, MultiBall, Pads and Switches Objects are vectors when the Object features sub-objects (when they feature more than one ball, slider, pad etc..). For example, the x coordinates of the balls in a MultiBall object are accessed as

- myball.x[0]
- myball.x[1]
- etc.

Now let's look at what happens to a vector when we use it in an expression and combine it with operators, singletons or other vectors.

Assume we have a MultiSlider object called **moo** and a MultiBall object called **bar**. Below we have listed some expressions involving vectors and the results they would produce. Assume **moo.x** contains {0.2 0.3 0.6} and **bar.x** contains {0.25 0.5}

- **moo.x \* 3**      Multiply all elements of **moo.x** by 3, returns {0.6 0.9 1.8}
- **moo.x + 2**      Add 2 to all elements of **moo.x**, returns {2.2 2.3 2.6}
- **moo.x>0.5**      Return a vector consisting of 1 or 0 depending on whether **moo.x** is greater than 0.5, returns {0. 0. 1.}
- **moo.x + bar.x**    Add all elements of two vectors. The size of the result is the smaller of the two input vectors. Returns {0.45 0.8}
- **moo.x[0]**         The first (index 0) element of **moo.x**, returns 0.2

- `bar.x[1]` The second element of bar.x, return 0.5
- `moo.x[0.5]` Interpolated value between bar.x[0] and bar.x[1], returns 0.3
- `{moo.x, bar.x}` Concatenates moo.x and bar.x, returns {0.2 0.3 0.6 0.25 0.5}
- `{moo.x[0],bar.x[1.5]}` Creates a new vector consisting of the first element of moo.x and the average of the second and third elements of bar.x, returns {0.2 0.625}

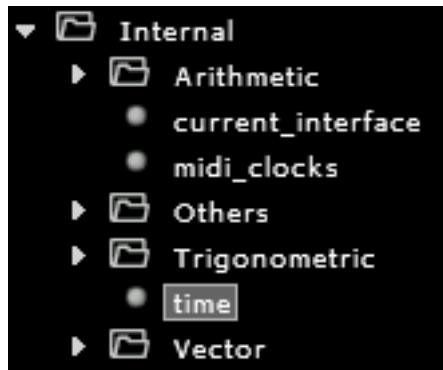
## 13.2 Lemur Internal clock

The Lemur works with a clock speed of **60 ticks per second**. This means that every **16 ms** all states of Objects, Expressions and Functions are evaluated. These 16 ms constitute one Lemur **frame**.

There are some special considerations, though, as the Lemur's brain tries to avoid unnecessary work. When it comes to deciding whether the value of a Variable is going to be transmitted via MIDI or OSC, the Lemur first checks if the Variable has changed since the last frame. If it hasn't changed, it's not going to be transmitted.

## 13.3 Built-in Time Variable

If you look at the Internal folder in the Project panel, you will see a pre-defined Variable called **time**.



The time Variable is a millisecond value you can use for creating time-varying behaviour in your Lemur. It represents the number of milliseconds since your Lemur was turned on and it resets to 0 every hour. The time value is intended to be manipulated by mathematical Operators, particularly by multiplication (\*), division (/), and by the modulo operator (% used to calculate the remainder of a division)

By multiplying time by a value greater than 1, you produce a sequence of values that increase faster than clock time. By multiplying time by a value less than 1, you will produce a sequence of values that increase more slowly than clock time. You can use the modulo Operator (%) to create a repeating sequence of values that resembles a sawtooth wave. For example `time % 1` produces a ramp from 0 to 999 that occurs over the course of a second.

Let's use a Monitor Objects to display the values of the following expressions:

**Frames** = `(time % 0.25) * 100`

**Hours** = floor(time/3600)

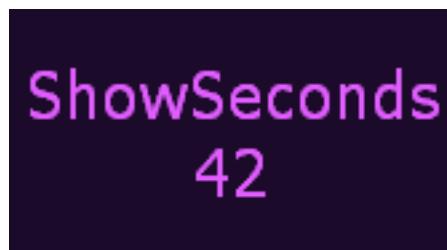
**Minutes** = floor(time/60) % 60

**Seconds** = floor(time % 60)

Create global Variables for each of these Expressions, then use these Variables in other Lemur Objects. For example, here is the Properties Browser view for a monitor Object that would display a global Seconds variable.



Even if you are not connected to a Lemur, the resulting Monitor Object will begin changing immediately after you enter a time-based variable.



Another interesting use of the time variable would be the generation of an LFO:

**LFO** = sin(time \* 2) \*0.5 + 0.5

You can display this in a SignalScope to see what it does.

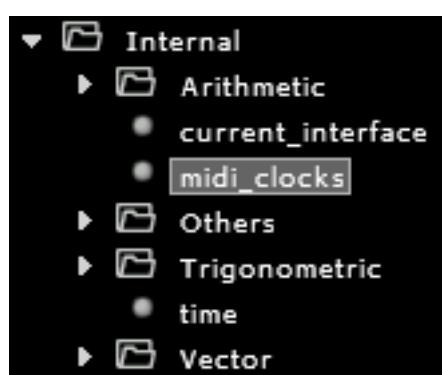
You can assign time-based variables to properties in Lemur Objects (such as Friction or the ADSR envelopes) to create Objects that change over time.

## 13.4 Built-in current\_interface Variable

The internal folder also features the **current\_interface** Variable, which simply gives you the Index of the currently displayed interface.

## 13.5 Built-in midi\_clocks Variable

The Internal folder also features the **midi\_clocks** variable:



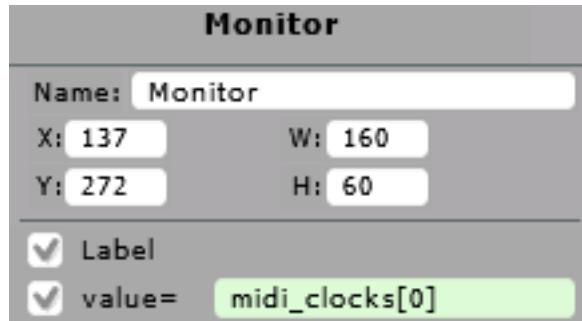
The midi\_clocks variable can sync to incoming MIDI Clock signal on the 8 MIDI Targets' inputs. Let's check it out with Ableton Live: make a connection between your Lemur's MIDI Target 0 Input and Live Sync Output:

JazzDaemon Targets			
	Inputs	Outputs	KbMouse
0	ordinateur-de-hugues-cla... : Daemon Output 0	none	none
1	none	none	none

MIDI Ports	Track	Sync	Remote
Output: IAC Driver (Bus IAC 2)	Off	Off	Off
Output: IAC Driver (Live Remote)	Off	Off	Off
Output: IAC Driver (Live Sync)	Off	Off	Off
Output: Daemon Output 0	Off	On	Off

Now use a Monitor Object to display the MIDI clock ticks on Target 0:



## 13.6 Arithmetic Functions

**abs(a):** absolute value

$$\text{abs}(-2) = 2$$

**ceil(a):** rounded value to higher integer

$$\text{ceil}(1.2) = 2$$

$$\text{ceil}(1.8) = 2$$

**clamp(a, min, max):** constrains a between min max

clamp(Fader.x, 0.2, 0.3) =

0.2 if Fader.x<=0.2

0.3 if Fader.x>=0.3

Fader.x otherwise

**floor(a):** rounded value to the lower integer

floor(1.2) = 1

floor(1.8) = 1

**max(a,b):** max of a and b

max(0.1, 0.8) = 0.8

**min(a,b):** min of a and b

min(0.1, 0.8) = 0.1

**pow(a,b):** a to the power of b

pow(2,4) = 16

**rand():** returns a random value between 0 and 1 at each frame (a Lemur frame is 16ms)

**range(a, min, max):** stretches a Variable that normally goes from 0 to 1, to range [min, max]

range(Fader.x, 10, 100) =

10 if Fader.x == 0

100 if Fader.x == 100

between 10 and 100 otherwise

**round(a):** rounded value to the closest integer

round(1.2) = 1

round(1.8) = 2

**sign(a):** -1 if  $a < 0$ , +1 if  $a \geq 0$

sign(-12) = -1

sign(0) = +1

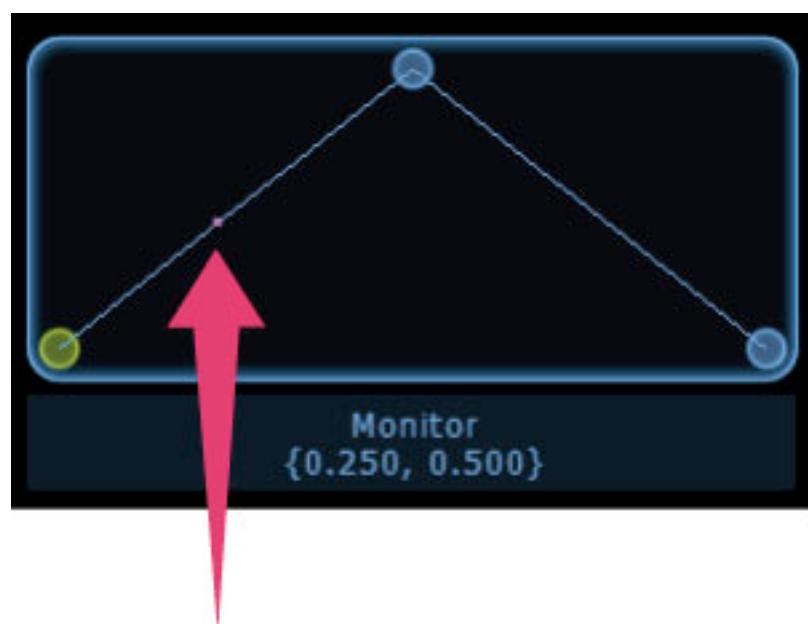
sign(12) = +1

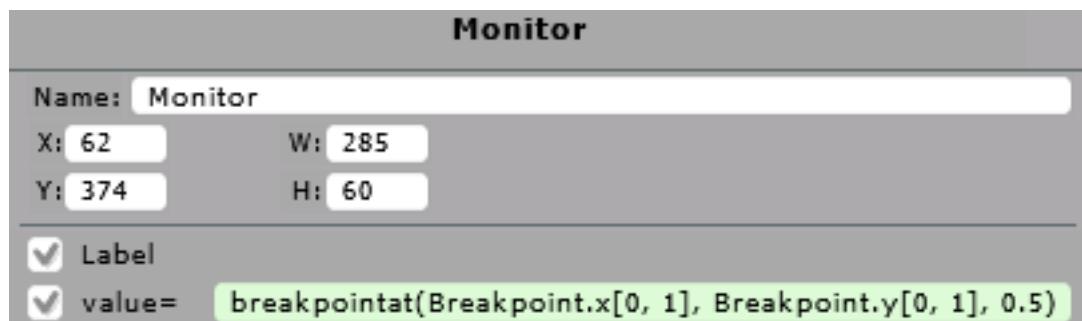
**sqrt(a):** square root of a

sqrt(64) = 8

## 13.7 Object-related Functions

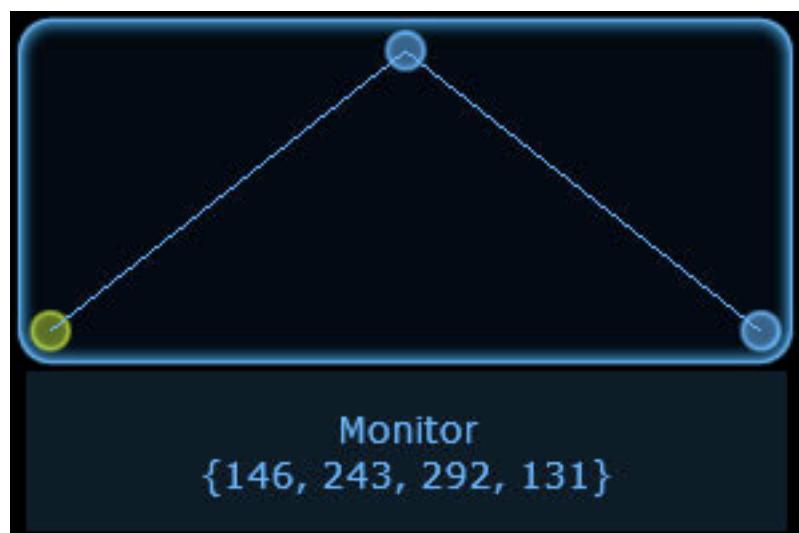
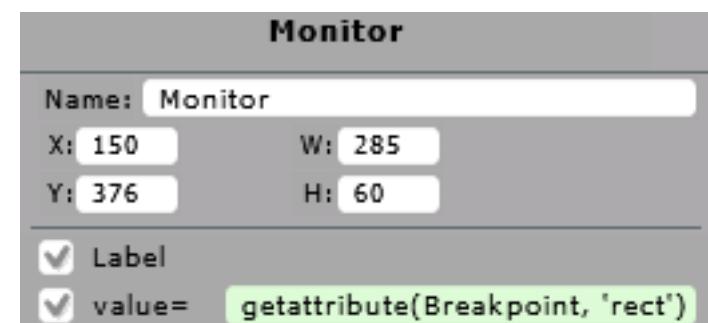
**breakpointat(x[],y[],phase):** returns the position of a point along a Breakpoint's segment





**findobject(name):** selects an Object within a tree structure (project, interface or container)

**getattribute(object, name):** returns the value of the Object's attribute 'name'



**getattributelist(object):** returns the Object's attribute list

**getexpression(object, name):** returns the value of the expression 'name'

**getfirst(object):** selects the first Object within a tree structure (project, interface or container)

**getnext(object):** selects the next Object within a tree structure (project, interface or container)

**getobjectrect(object):** returns the Object's position and dimensions as {X,Y,W,H}

**getparent(object):** returns an Object's Parent Target (next higher hierarchy level)

**RGB(r,g,b):** converts 3 RGB primary color indices (Red, Green, Blue) into a single value that the parser understands

The RGB(r,g,b) function performs the operation  $((R \times 2^{16}) + (G \times 2^8) + B)$ , where R, G and B are values between 0 and 127.

The RGB(r,g,b) function is used to obtain a value for an Object's color attribute the value of the color attribute can range from 0 to 8355711

**selectinterface(index):** displays the selected Interface

**selecttab(object, index):** displays the selected tab (Container Object)

**setattribute(object, name, value):** sets an Object's attribute 'name' to the desired value

**setexpression(object, name, value):** sets an Objects's expression to the given value

**setobjectrect(object, rect[]):** sets an Object's position and dimensions to {X,Y,W,H}

**show(object,state):** displays (state=1) or hides (state=0) an object

## 13.8 Scripting Output Functions

<b>ctlout(target,ctrl,val,chan):</b>	outputs a Control Change MIDI message with specified Target, Controller number, value and channel.
<b>keycomboout(target,ctrl,alt,shift,key):</b>	outputs a simultaneous key strokes combination to the defined KbMouse Target.
<b>keyout(target,key,state):</b>	outputs a single key stroke to the defined KbMouse Target.
<b>midfout(target,msg[]):</b>	outputs the defined MIDI message to the defined MIDI Target.
<b>mousebutton(target,state):</b>	outputs a left mouse button click to the specified KbMouse Target.
<b>mousemove(target,x,y):</b>	outputs a x and y mouse movement to the specified KbMouse Target.
<b>noteout(target,note,vel,chan):</b>	outputs a Note On MIDI message with specified Target, Controller number, value and channel.
<b>oscout(target,address,args[]):</b>	outputs the defined OSC command to the specified address and Target.

## 13.9 Vectorial Functions

**arrayToString(array[]):** converts an array to a string of values

**diff(a):** returns the difference between the consecutive elements of an array

if  $a = \{w, x, y, z\}$

$\text{diff}(a) = \{w, x-w, y-x, z-y\}$

Example:

$\text{diff}(\{1, 2, 3, 4, 5, 6, 7, 8\}) = \{1, 1, 1, 1, 1, 1, 1\}$

**fill(a,value,size):** returns a vector with "size" items

the vector is filled from the left with n items = "value"

the number of items set to "value" depends on the a argument : a threshold between 0 and 1

when  $a = 0$ , there's no filling at all, and the vector is full of zeros

when  $a = 1$ , the vector is completely filled with items = value

when  $a = 0.5$ , the vector is half filled

Examples:

$\text{fill}(1, 0.524, 4) = \{0.524, 0.524, 0.524, 0.524\}$

$\text{fill}(0, 0.524, 4) = \{0, 0, 0, 0\}$

$\text{fill}(0.5, 0.524, 4) = \{0.524, 0.524, 0, 0\}$

**firstof(a):** returns the position of the first non-null item in a vector

This is often used with switches in radio mode : firstof(x) returns the position of the enabled switch in the matrix

Examples:

firstof({0,0,1}) = 2

firstof({1,0,0}) = 0

if the vector only contains null items, the function returns the size of the vector

firstof({0,0,0}) = 3

**interlace(a,b):** interlace 2 vectors to form a single vector

if a={1,2,3}, b={5,6,7}

interlace(a,b)={1,5,2,6,3,7}

**interlace3(a,b,c):** interlace 3 vectors to form a single vector

if a={1,2,3}, b={5,6,7} and c={8,9,10}

returns {1,5,8,2,6,9,3,7,10}

**nonnull(array):** returns the positions of non-null items in an array

nonnull({0,0,1,0})=2

**replace(a,b,position):** takes an array, and replace the items starting at "position" with b (b can be an array or a singleton)

Examples:

replace({0,0,0,0},{12,2},0) = {12, 2, 0, 0}

replace({0,0,0,0},0.15,3) = {0, 0, 0, 0.15}

If position is not an integer, the parser converts it using the floor function (next lower integer)

**set(a,value,position):** takes an array, and change the item at "position" to "value" (position can be an array or a singleton)

Examples:

`set({0,0,0,0}, 12, 0) = {12, 0, 0, 0}`

`set({0,0,0,0}, 12, {0,2}) = {12, 0, 12, 0}`

If position is not an integer, the parser converts it using the floor function (next lower integer)

**sizeof(a):** returns the size of a vector

**stretch(a,size):** stretches a value or a range and returns a vector

If a is a single value (or a vector with a single item), the function returns a vector with "size" items containing that value

`stretch(0, 12) = {0,0,0,0,0,0,0,0,0,0,0,0}`

after that, you could do :

`set(stretch(0, 12), 1, time%12), this returns:`

`{1,0,0,0,0,0,0,0,0,0} when time<1`

`{0,1,0,0,0,0,0,0,0,0} when time<2`

`{0,0,1,0,0,0,0,0,0,0} when time<3 etc..`

If a is a vector with more than one value, the function stretches the range over "size" items :

`stretch({1,4}, 4) = {1,2,3,4}`

`stretch({0,10},11) = {0,1,2,3,4,5,6,7,8,9,10,11}`

**subarray(array,start,length):** chops an array into a defined sub-array

`if a={1,2,3,4,5,6,7,8}`

`subarray(a,2,3)={3,4,5}`

**sumof(a):** sums all the items of an array

**wrap(a,size):** if `a={1,2,3,4,5,6,7,8}`, `wrap(a,5)={1,2,3,4,5}`

## 13.10 Trigonometric functions

**acos, asin, atan, cos, sin, tan, log, log10, exp** are the same as their mathematical counterparts.

**angle(x,y):** returns the angle in radians formed by a vector of coordinates (x,y) to the positive x axis

Example:

Create a RingArea

Create a variable inside : a = angle(x-0.5, y-0.5)

this returns the angle position of the ball from the centre of the RingArea

**norm(x, y):** length of a vector of coordinates (x,y)

Example :

create a Multiball with 2 balls

variable inside : distance = norm(x[1]-x[0], y[1]-y[0])

this returns the distance between ball 0 and ball 1

## 13.11 Operators

### Arithmetic

+	add
-	subtract
*	multiply
/	divide
%	modulo
++	increment
--	decrement

## Comparison

>	greater than
<	smaller than
>=	greater or equal to
<=	smaller or equal to
==	equal to

## Logic

When using logic and conditions in the parser, (not 0) is true and 0 is false.

**!a**                    not a

1 when a = 0

0 when a!=0

a!=b

1 if a!=b (i.e a=2, b=4)

0 if a==b

**a&&b**                    logical AND

1 && 1 = 1

1 && 0 = 0

**a<b**                    1 if a < b

0 otherwise

**a>b**                    1 if a>b

0 otherwise

**a==b**                    1 if a==b

0 otherwise

**a>=b**                    1 if  $a \geq b$

0 otherwise

**a<=b**                    1 if  $a \leq b$

0 otherwise

**a || b**                    logical OR

$1 || 0 = 1$

$1 || 1 = 1$

**a?b:c**                    conditional statement, which translates as: "if (a is true) then b else c "

Example:

Creating an Expression "e" in a MultiBall and setting its value to *Pad.x?x:y* will mean that :

if  $\text{Pad.x} \neq 0$  (i.e Pad is pressed),  $e=x$

if  $\text{Pad.x} == 0$  (i.e Pad is not pressed),  $e=y$

You could for example map the e Expression to a Control Change mapped to the x axis when the pad is pressed, and y axis when the pad is released

## Assignment

Assignment operators allow the same variable name to contain different values at different times during the script execution.

=                            gets set to

+=                            add and assign

-=                            subtract and assign

/=                            divide and assign

\*=                            multiply and assign

## Bitwise

Bitwise operators allow operations on array patterns at the level of their individual elements.

& binary AND

| binary OR

<< left shift

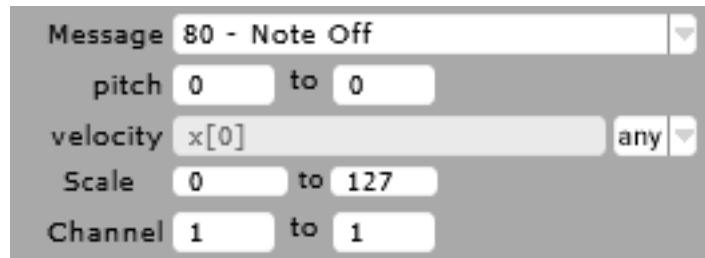
>> right shift

# Chapter 14 - MIDI Message Reference

The MIDI section of the JazzEditor Mapping Panel offers a wide range of MIDI messages your Objects' Variables can directly be assigned to. This Chapter provide standard definitions of these messages as given by the MIDI specifications. Of course it is up to you to use the various messages to your liking. In a world of flexible and modular software, MIDI messages tend to lose their original meanings. In modular software like Reaktor or Max/MSP you can use any MIDI message to trigger or change anything you want.

## 14.1 Note Off

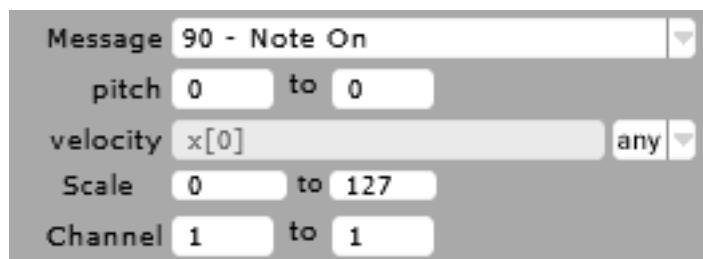
The Note Off message is sent when a key is released.



- pitch: Dial in the MIDI note number. If the value is a vector the second field will be extrapolated automatically. You can always override this setting by putting in your own value.
- velocity: The Variable chosen from the Variable menu that is converted to velocity values is shown here.
- Scale: Type in the desired target scaling for the MIDI messages. Possible values range from 0 to 127.
- Channel: Dial in the MIDI channel. If the value is a vector the second field will be extrapolated automatically. You can always override this setting by putting in your own value. Possible values range from 0 to 16.

## 14.2 Note On

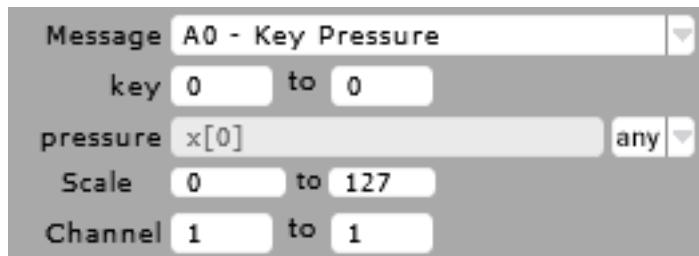
The Note On message is sent when a key is depressed.



- pitch: Dial in the MIDI note number. If the value is a vector the second field will be extrapolated automatically. You can always override this setting by putting in your own value.
- velocity: The Variable chosen from the Variable menu that is converted to velocity values is shown here.
- Scale: Type in the desired target scaling for the MIDI messages. Possible values range from 0 to 127.
- Channel: Dial in the MIDI channel. If the value is a vector the second field will be extrapolated automatically. You can always override this setting by putting in your own value. Possible values range from 0 to 16.

### 14.3 Key Pressure (Polyphonic Aftertouch)

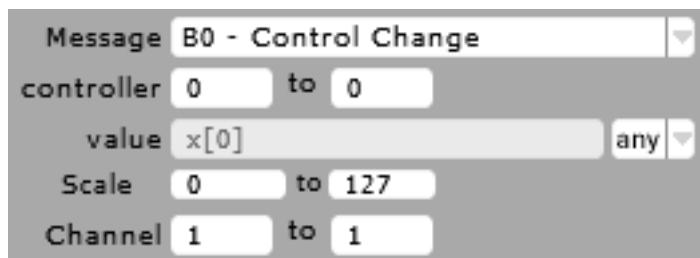
The Key Pressure, also called polyphonic aftertouch, gives the keyboarder a continuous controller for every key he had on his keyboard. It is used via pressing down on the different keys. This works great for spreading out vector values to multiple key pressure values.



- key: Dial in the MIDI note number. If the value is a vector the second field will be extrapolated automatically. You can always override this setting by putting in your own value.
- pressure: The Variable chosen from the Variable menu that is converted to key pressure values is shown here.
- Scale: Type in the desired target scaling for the MIDI messages. Possible values range from 0 to 127.
- Channel: Dial in the MIDI channel. If the value is a vector the second field will be extrapolated automatically. You can always override this setting by putting in your own value. Possible values range from 0 to 16.

## 14.4 Control Change

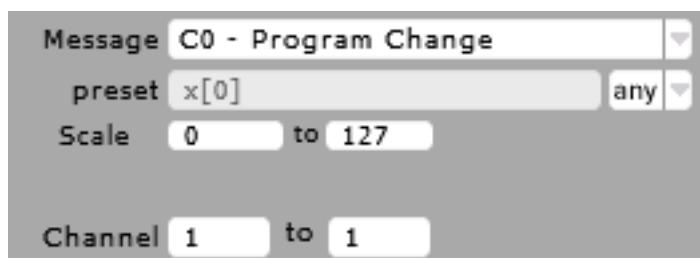
The Control Change data is used to produce pseudo-continuous data (ranging from 0 to 127).



- controller: Dial in the MIDI controller number. If the value is a vector the second field will be extrapolated automatically. You can always override this setting by putting in your own value. Possible values range from 0 to 127
- value: The Variable chosen from the Variable menu that is converted to the controller values is shown here.
- Scale: Type in the desired target scaling for the MIDI messages. Possible values range from 0 to 127.
- Channel: Dial in the MIDI channel. If the value is a vector the second field will be extrapolated automatically. Possible values range from 0 to 16.

## 14.5 Program Change

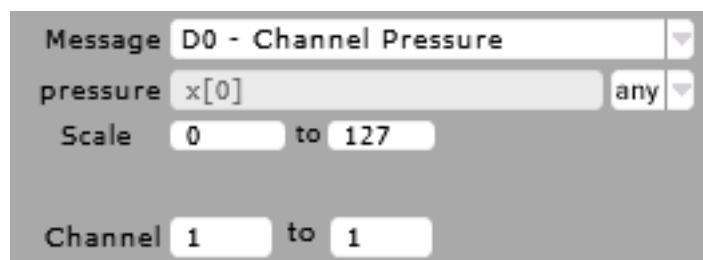
Used to change the patch number of synthesizers.



- preset: The variable chosen from the Variable menu that is converted to program change values is shown here.
- Scale: Type in the desired target scaling for the MIDI messages. Possible values range from 0 to 127.
- Channel: Dial in the MIDI channel. If the value is a vector the second field will be extrapolated automatically. Possible values range from 0 to 16.

## 14.6 Channel Pressure

Another continuous controller used on a per MIDI channel basis.



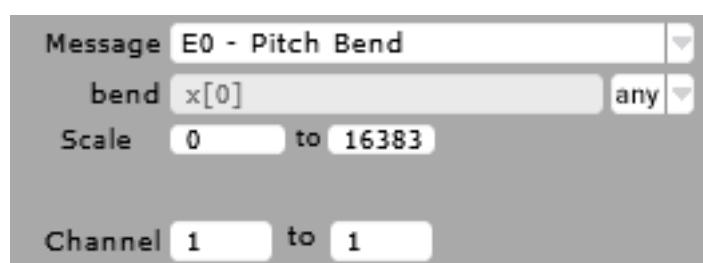
**pressure:** The Variable chosen from the Variable menu that is converted to channel pressure values is shown here.

**Scale:** Type in the desired target scaling for the MIDI messages. Possible values range from 0 to 127.

**Channel:** Dial in the MIDI channel. If the value is a vector the second field will be extrapolated automatically. Possible values range from 0 to 16.

## 14.7 Pitch Bend

Used to indicate a change in the pitch wheel. It is a 14 bit value providing a higher resolution than control changes.



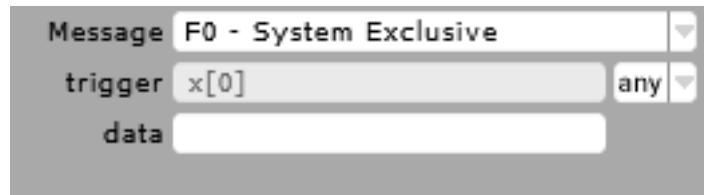
**bend:** The Variable chosen from the Variable menu that is converted to program change values is shown here.

**Scale:** Type in the desired target scaling for the MIDI messages. Note that pitch bend values range from 0 to 16383.

**Channel:** Dial in the MIDI channel. If the value is a vector the second field will be extrapolated automatically. Possible values range from 0 to 16.

## 14.8 System Exclusive

Used for custom data that is not covered by normal MIDI messages. The first part is usually a 7 bit value representing an ID. If the device or software recognizes the ID as its own, it will listen to the rest of the message. Otherwise, the message will be ignored.



Trigger: The Variable chosen from the Variable menu that is converted to SysEx messages is shown here.

Data: Type in the desired SysEx string you want to send. Please have a look at the MIDI Target's documentation for details about possible SysEx messages.

## 14.9 Song Position

A 14 bit value that holds the number of MIDI beats (1 beat= six MIDI Timing Tick) since the start of the song.

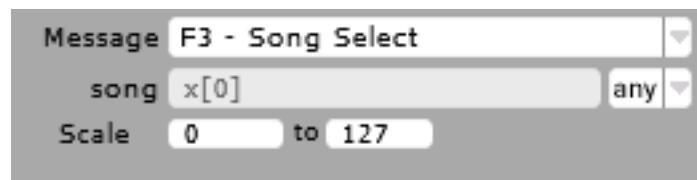


Position: The Variable chosen from the Variable menu that is converted to song position values is shown here.

Scale: Type in the desired target scaling for the position pointer. Note that song position pointer values range from 0 to 16383.

## 14.10 Song Select

The Song Select specifies which sequence or song is to be played.

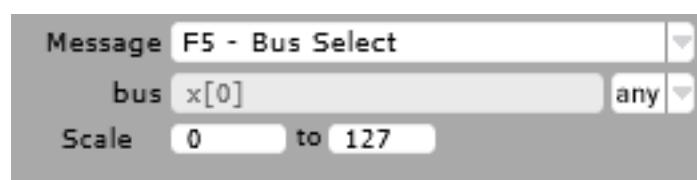


Song: The Variable chosen from the Variable menu that is converted to song number values is shown here.

Scale: Type in the desired target scaling for the song number. Possible values range from 0 to 127.

## 14.11 Bus Select

The Bus Select message specifies to which MIDI output further data should be sent.



Bus: The Variable chosen from the Variable menu that is converted to bus number values is shown here.

Scale: Type in the desired target scaling for the bus number. Possible values range from 0 to 127.

## 14.12 Tune Request

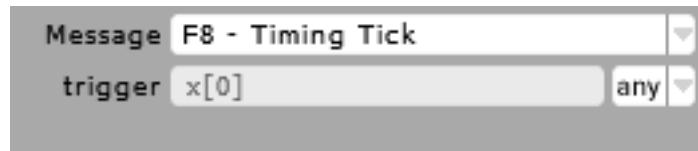
Used to trigger the tuning of oscillators in analog synthesizers.



Trigger: The Variable chosen from the Variable menu that is used to trigger tune request messages is shown here.

## 14.13 Timing Tick

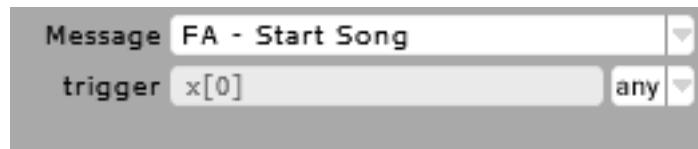
Sent 24 times per quarter note for synchronization purposes. Build your own MIDI clock.



Trigger: The Variable chosen from the Variable menu that is used to trigger timing tick messages is shown here.

## 14.14 Start Song

Used to start the song or sequence from the beginning.



Trigger: The Variable chosen from the Variable menu that used to trigger start song messages is shown here.

## 14.15 Continue Song

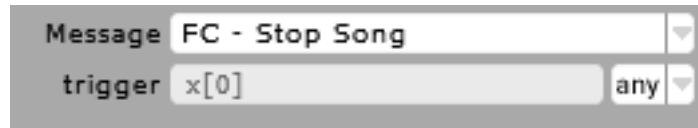
Used to start the song or sequence from where it was stopped.



Trigger: The Variable chosen from the Variable menu that is used to trigger tune request messages is shown here.

## 14.16 Stop Song

Used to stop the song or sequence.



Trigger: The Variable chosen from the Variable menu that is used to trigger stop song messages is shown here.

## 14.17 Active Sensing

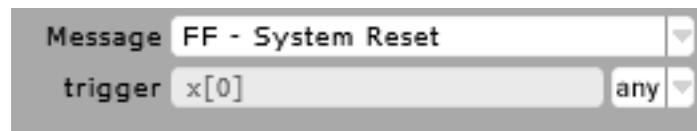
When initially sent, the receiver will expect to receive another Active Sensing message each 300ms (max), or it will assume that the connection has been terminated. At termination, the receiver will turn off all voices and return to normal (non-active sensing) operation.



Trigger: The Variable chosen from the Variable menu that is used to trigger active sensing messages is shown here.

## 14.18 System Reset

Resets all receivers in the MIDI chain to power-up status.



Trigger: The Variable chosen from the Variable menu that is used to trigger system reset messages is shown here.

# Appendix I - Network Configuration Guide

The Lemur control surface uses the Ethernet protocol to connect and communicate with other devices on the network: Computers, routers, DHCP servers...

Generally, a device can have multiple network adapters, like Ethernet, WIFI, Airport, USB Network Adapters, Firewire and others. Each adapter can have its own IP settings and it is highly recommended to set different sub-networks to avoid network problems.

## AI.1 Glossary

**IP Address** a unique sequence of 4 numbers which identify each device on a network. An IP address can be setup in static mode or attributed dynamically by a DHCP server. An IP address is composed by 4 numbers between 0 and 255 as follows: 192.168.2.2 or 10.0.0.35.

**Network Mask** an IP-like sequence which defines a sub-network of compatible IP addresses. Most of the time we use 255.255.255.0 or 255.255.0.0, which means that only the 3 or 2 first IP numbers should be equal to be compatible.

**Gateway** the IP address of the machine giving you access to IPs which are not included in the sub-network. In brief, a Gateway enables you to gain access to the Internet from an intranet.

**MAC Address** a sequence of 6 numbers identifying each network adapter on a network. The MAC Address is unique and constant for each device.

## AI.2 Direct Connection

Using a simple crossover RJ45 cable, you can directly connect your Lemur to any computer.

### AI.2.1 Computer configured as a DHCP Server – Connection Sharing

If your computer is configured as a DHCP Server, it will attribute dynamically an IP address to the Lemur. Most of the time your computer forces its IP address to 169.254.X.1 and the Lemur will obtain an IP like 169.254.X.X.

### AI.2.2 Computer and Lemur using DHCP

When the Lemur and the computer are set up as DHCP clients, they both wait for a DHCP server until the OS gives a default IP address like 169.254.X.X.

### AI.2.3 Computer and Lemur using Static IP

The most common way to configure a network without a DHCP server is to set up each device with different but compatible IP's using the same network mask. Most people use IP addresses like 192.168.0.X or 10.0.0.X.

Eventually, when using connection sharing, the computer can share its Internet connection and give the Lemur access to another sub-network by using the Gateway Address.

## AI.3 Computers and Lemur connected to a Network

### AI.3.1 Network with a DHCP Server

Most network sharing devices allow multiple connections and manage the attribution of IP addresses through the DHCP protocol. These devices include WIFI/Ethernet Routers, Airport Base Stations and ADSL/Cable Modems with routers.

You only need to configure all your devices - including the Lemur - to use the DHCP mode.

### AI.3.2 Network without DHCP Server

If you have multiple devices connected through a hub or switch, you may have to configure them individually with a unique, static IP sharing the same network mask.

Use IPs like: 192.168.0.X or 10.0.0.X

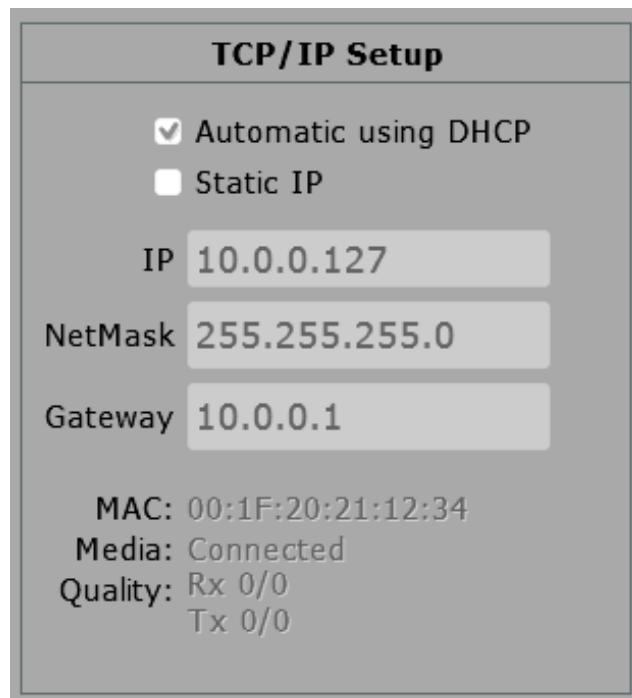
Mask: 255.255.0.0

## AI.4 Lemur Network Setup

Open the Network Settings via the Settings button, the left-most hardware button above the Lemur's touchscreen.

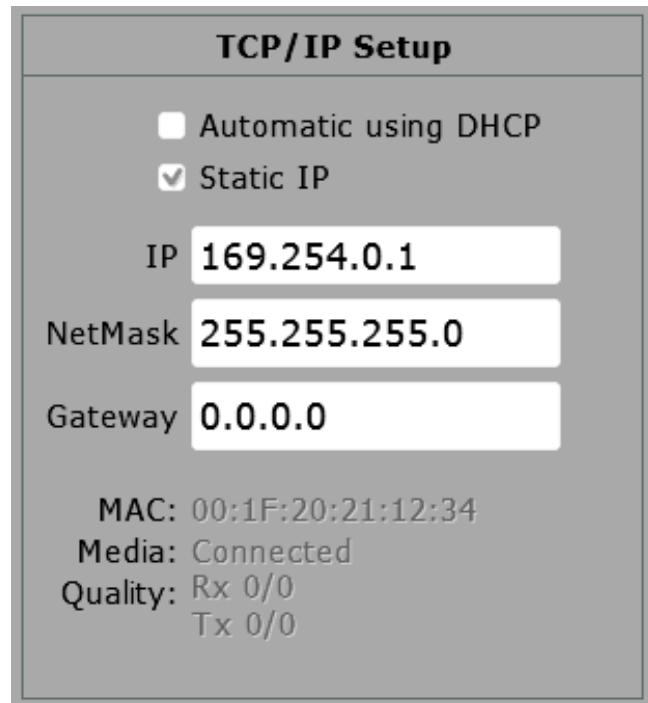
DHCP IP Configuration:

- Select "Automatic using DHCP".
- If a DHCP server is available, the Lemur will obtain a DHCP IP address automatically. If there isn't a DHCP server available, the Lemur will choose its own address.



**Static IP** Configuration:

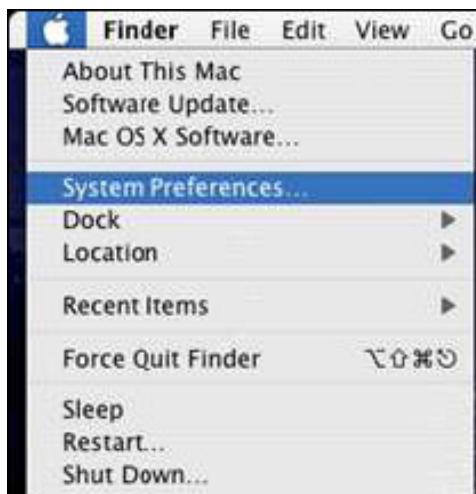
- Select “Static IP”.
- Enter assigned static IP address into the IP field.
- Enter NetMask (255.255.255.0)



## AI.5 MacOS Network Setup

### AI.5.1 Ethernet Interface Static IP Configuration using Mac OS X

- Go to the **Apple menu** and select **System Preferences**.



- Under **Internet and Network**, select **Network**.

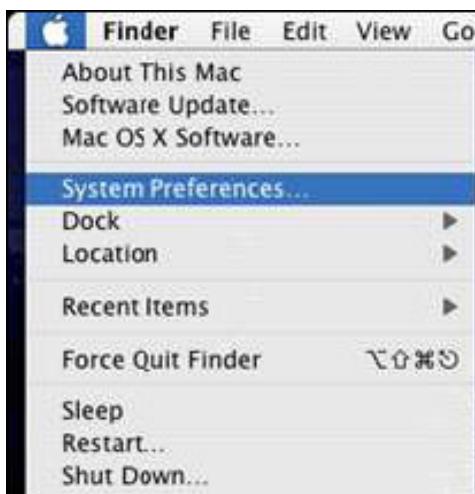




- Change the **Show** pop-up option to **Built-in Ethernet**.
- Under **Location**, choose **Automatic** (or name the new connection).
- Under **Show**, Select **Built-in Ethernet**.
- Change to the **TCP/IP** tab.
- Set **Configure IPv4** to **Manually**.
- Enter assigned static **IP Address**.
- Enter **Subnet Mask** **255.255.255.0**
- Click the **Apply** Now button, and **quit** the System Preferences application.

## A1.5.2 DHCP Configuration using Mac OS X

- Go to the Apple menu and select System Preferences.



- Under Internet and Network, select Network.





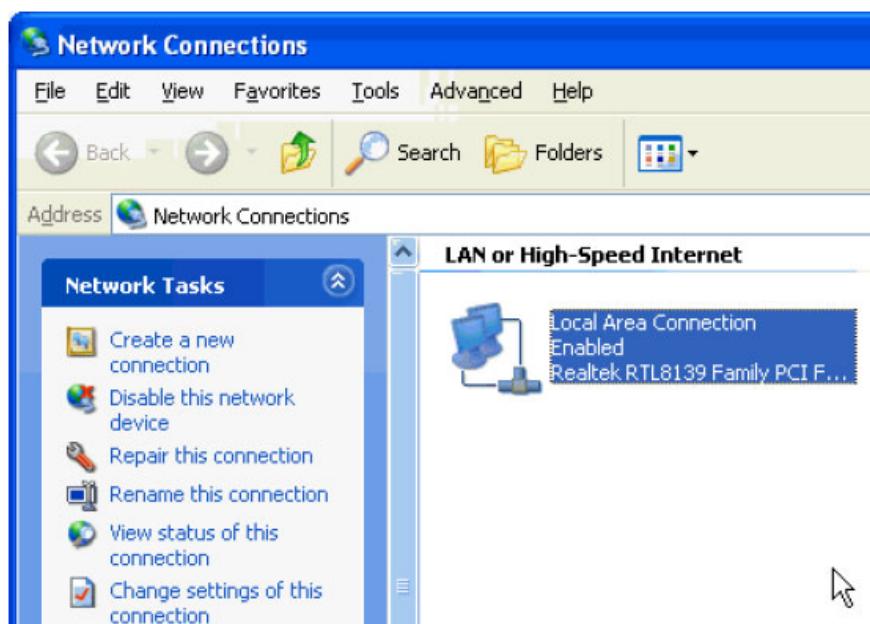
- Change the **Show** pop-up option to **Built-in Ethernet**.
- Under **Location**, choose **Automatic** (or give this connection a name).
- Under **Show**, Select **Built-in Ethernet**.
- Change to the **TCP/IP** tab.
- Set **Configure IPv4** to **Using DHCP**.
- Click the **Apply Now** button, and **quit** the System Preferences application.

## AI.6 Windows XP Network Setup

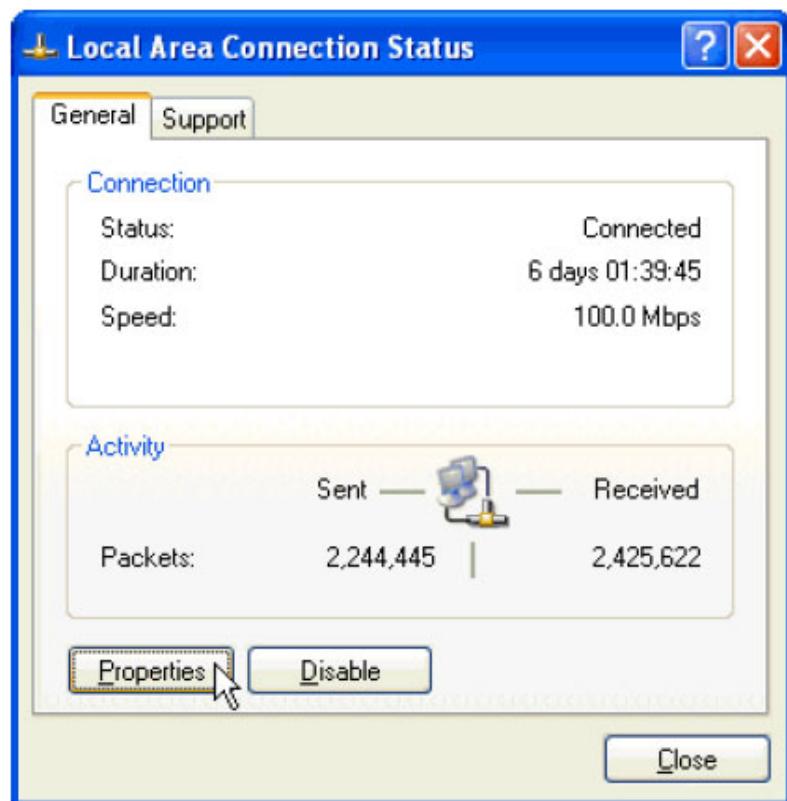
- On your Desktop, right-click on the **My Network Places** icon and select **Properties**.



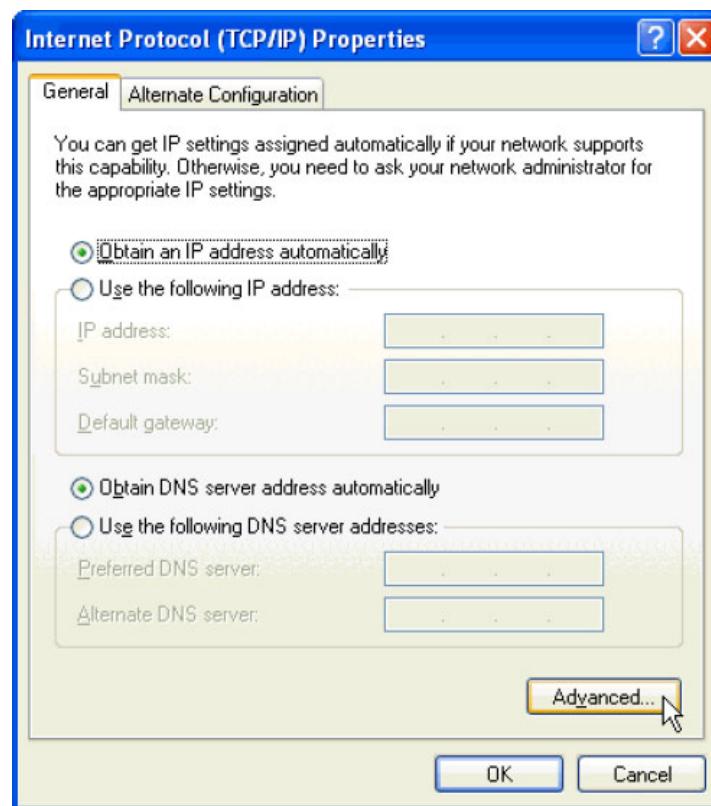
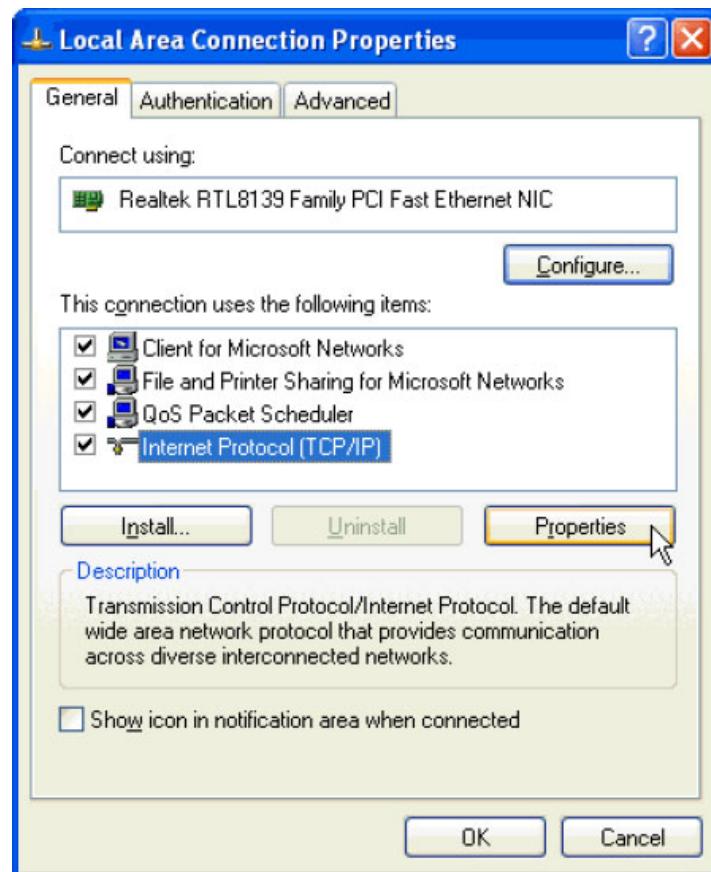
- Select your **LAN connection** and double-click on it.



- Click on the **Properties** button.

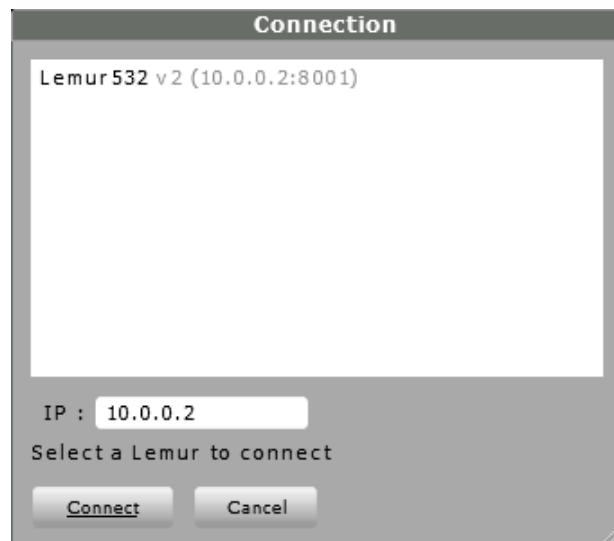


- Select **Internet Protocol (TCP/IP)** and click the **Properties** button.



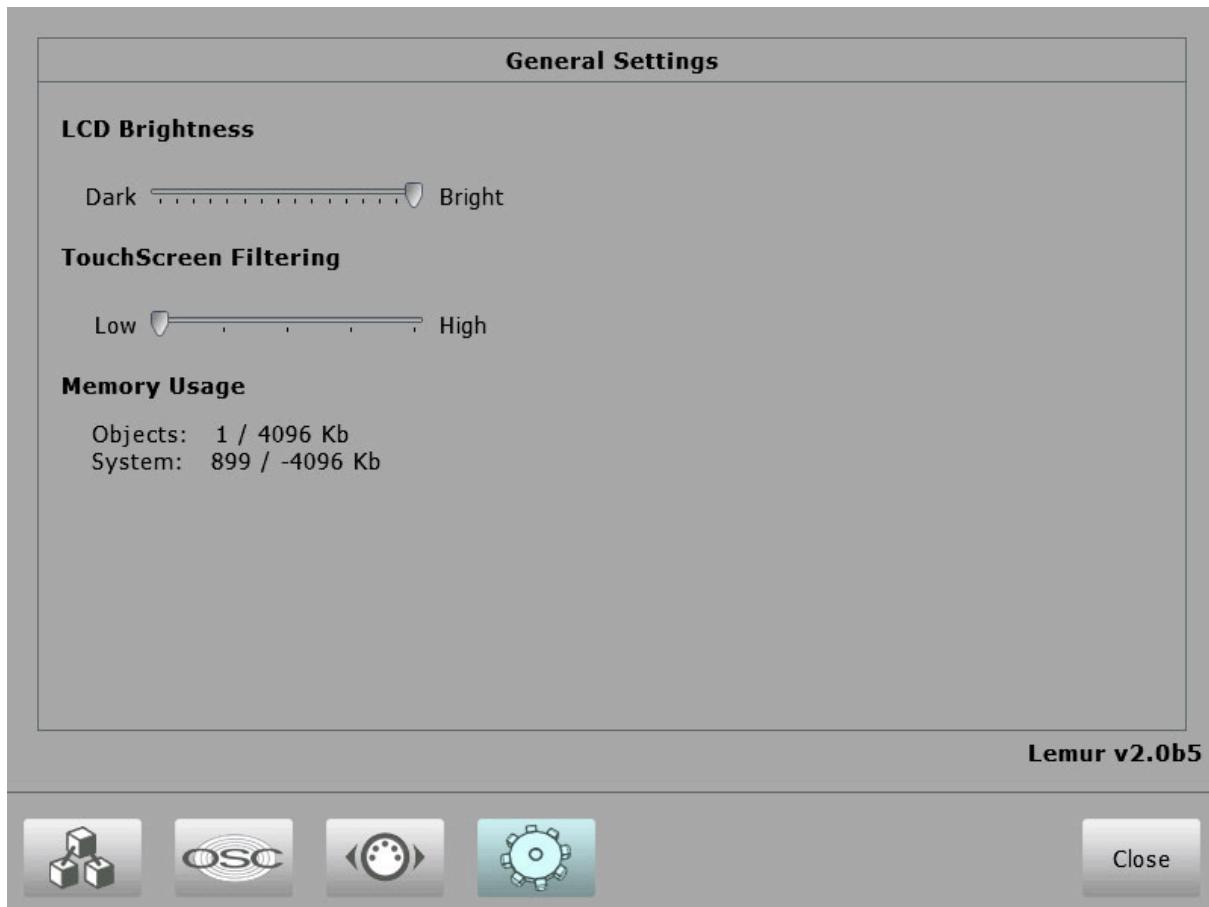
- Select **Obtain an IP address automatically** to use DHCP configuration or select **Use the following IP address** to use static IP configuration. Enter assigned static IP address and Subnet Mask.
- Click **OK** to confirm changes.

To confirm that your network setup is correct, power on the Lemur and open the JazzEditor. Click the Connect button. If your Network settings are correct, your Lemur should appear on the list.



## Appendix II - Hardware General Settings

The leftmost button on your Lemur opens the Hardware Settings. The fourth button at the bottom of the page lets you access the General Settings page:



The page features two sliders for controlling your Lemur's screen hardware:

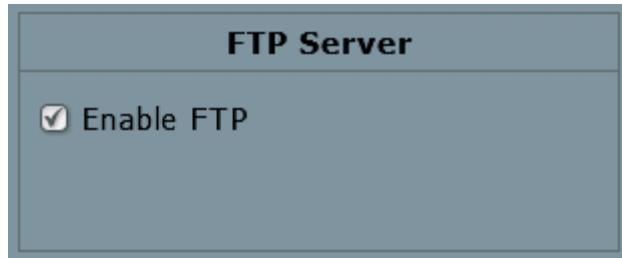
- **LCD Brightness** controls the brightness of your Lemur's screen. It may be useful to dim the display if you want to work under dark stage conditions where too much light may not be desirable.
- **TouchScreen Filtering** is for controlling the sensitivity of the Lemur's touch screen. If you experience multiple triggering or other touch artefacts, try a higher Filtering level.

## Appendix III - FTP Server

Yep, the Lemur runs an FTP server. This can be helpful for uploading large numbers of Projects to the Lemur without having to channel them through the JazzEditor.

It can also come in very handy for retrieving Projects that you lost due to a mishap with the JazzEditor, e.g. if you accidentally closed it without saving. Simply save the Project to the internal memory of the Lemur, download it via FTP to your computer and everything is cool again. As we don't want everybody on the Network to log into our Lemur and fill it up with trash, the FTP server is deactivated by default.

Push the Settings button on your Lemur and you are presented with various Setup options. Touch the Enable FTP flag to activate the FTP server.



Now you can use your favourite **FTP client** to connect to the Lemur on its IP address. A web browser also does the job:



## Appendix IV - Shortcuts

New Project	Command/ Control+N
Save Project	Command/ Control+S
Import Library	Command/ Control+I
Copy	Command/ Control+C
Paste	Command/ Control+V
Redo	Command/ Control+Y
Lemur Synchro	Command/ Control+R
Lemur Settings	Alt+S
Create Object	Command/ Control+Shift+O
Create Custom MIDI	Command/ Control+Shift+M
Open Project	Command/ Control+O
Save Project As	Command/ Control+ Shift+S
Export Module	Command/ Control+E
Cut	Command/ Control+X
Undo	Command/ Control+Z
MIDI Map	Alt+M
Lemur Connection	Alt+C
Create Interface	Command/ Control+Shift+I
Create Expression	Command/ Control+Shift+E
Create Custom MIDI	Command/ Control+Shift+M
Create Script	Command/ Control+Shift+X
Toggle to Run mode	Hold E

## Appendix V - Warranty

### Warranty and Liability

Please check the delivered products for obvious defects. By opening the original packaging or downloading the product you accept all copyrights as well as the warranty restrictions. Should you find any defect of the merchandise during the one-year warranty period, JAZZMUTANT will - at its sole discretion - either replace the merchandise or remove any defects. Should we be unable to do so within an acceptable period of time, you are entitled to cancel your order. We are liable for the culpable breach of fundamental contractual obligations (so-called cardinal obligations) and the absence of any promised qualities. Otherwise, we are liable only within the limits set by applicable liability laws or - for whatever legal reason – when any damages were caused with intent or gross negligence by us or one of our affiliates. Neither our affiliates nor we are liable in any case not described above. In case of culpable breach of cardinal obligations without intent or negligence on our part, our affiliates or we will be liable only to the extent of typically foreseeable damages and not for damage not incurred to the delivered merchandise itself. The risk of loss is transferred to you once the merchandise has been sent to you. This

also applies with part deliveries. If you are buying software via electronic delivery (Instant Delivery), we carry the risk of loss until the software has been fully transferred to your computer.

### Your Rights

The customer's rights concerning refunds of all or part of a payment are valid only when such rights were legally confirmed or were accepted by us in writing. The orderer's right to retention is valid only provided claims are based on the same contract. The preferred method of written communication is email and is held to be equal in status to letters transmitted through the postal service. The term "correspondence" below is taken to include email. Deadlines are taken, insofar as is not otherwise mutually agreed, to always mean the date of receipt by the recipient. As far as email is concerned, the date of receipt is considered to be the date of dispatch. If there is disagreement about the date of dispatch, the time of receipt is taken as the receipt of the message on the target email server.

### Applicable Laws and Court of Jurisdiction

The law of France applies in exclusion of UN buying law, even when the order is placed in another country or the order is to be delivered into another country. If you are resident outside France, or if your usual place of abode is outside France, the court of jurisdiction for all claims in relation to your order is in Bordeaux, France. In the event of a legal dispute, the official address is JAZZMUTANT, 107, cours Balguerie Stuttenberg 33300 BORDEAUX, France.