OSC

Overview

Open Sound Control is a protocol for sending control messages across networks. It was developed by Adrian Freed and Matt Wright (working with David Wessel) at CNMAT in Berkeley in 1996 as a replacement or complement to MIDI. Acceptance was slow at first, (few people saw the point) but it gradually picked up steam as it was used in various cutting edge experimental devices. OSC has many advantages over MIDI, but it has two primary features: it uses standard networking hardware and it is, as advertised, open–we do not need to force our communications to fit the MIDI keyboard paradigm.

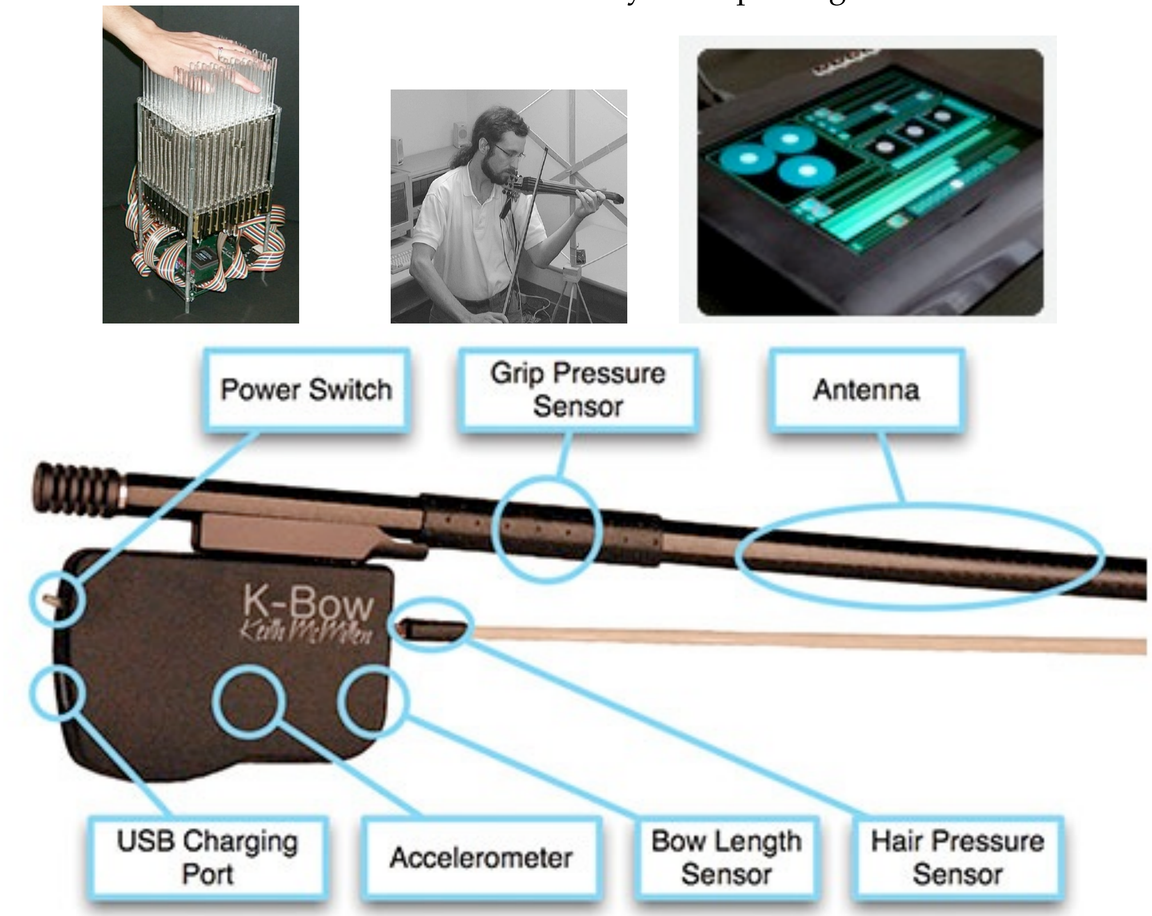


Figure 1 Some examples of OSC devices.

OSC can be used in many network schemes, but most current systems use Ethernet or wireless systems running UDP/IP. Every computer has one or the other network connection, and normal network traffic will not interfere with OSC messages. UDP (User Datagram Protocol) differs from the more common TCP in that it does not require handshaking between client and host or resending packets when errors are detected. This is a direct tradeoff of data integrity for speed. In other words, an occasional command may get lost, but the others will arrive on time. In practical use, messages are only lost on high traffic networks. Most performance setups are very simple, with a direct wire connection or a short wireless link. In these situations simple commands such as button1 on, button1 off, will be adequate. With noisier networks or more critical applications such as remote control of speaker systems the entire sate of the system is constantly transmitted. There are other approaches, but these are the ones you will see most often.

The typical Max application will consist of an OSC controller setting parameters of a Max patch or a Max Patch sending commands to an OSC controllable device. The actual commands in use will depend on the design of the external device or controller. The command list will be provided in the device documentation. (Devices or apps that are programmable usually allow custom commands.)

Network Connections
Communications protocols define devices as either host or client. In most cases, the computer is the host and the OSC gadget is the client, but there are systems that run the other way. This nomenclature is left over from the days when one host computer could connect to many client terminals but a client could only have one host. Now the situation is more complex–clients may talk to each other and there might be more than one host. Sometimes both client and host are programs running simultaneously on the same machine, and the names are just labels in the connection dialog.

Setting up a physical network is beyond the scope of this paper, but the components are easily available. In the simplest case, the two devices are connected by a CAT5 (Ethernet) cable. Occasionally, this must be a crossover cable[1], which has the plugs wired differently at each end. They are hard to find but available. In a more complex setup, you may want to use an Ethernet switch as a hub between the devices. This has the advantage that you can connect several devices and maintain internet access as well.

Simple switches cannot assign IP addresses, so the switch may need to connect to an internet router to acquire addresses. This is not necessary if the host is a Mac, as Macs have the ability to set up an ad hoc local area network (LAN) if there is no router available. (You can also set up server software on a Mac or PC to supply addresses, but that is way more work than it is worth.)

The next step in complexity is a router, which can be wireless or not. A router need not be connected to the internet–just follow the instructions for setting up a LAN. My best experiences have been with a wireless LAN with no internet connection. Apple AirPort offers this type of solution and is easy to set up. You can read all IP addresses from the utility program. Another Macintosh trick is to set up a temporary wireless LAN directly from the computer. This is handy in a pinch, but the IP addresses have to be set up each time the connection is established. The Airport solution will keep the same IPs.

[1] If you get one, label it, because it looks just like all of the other Ethernet cables, but won't work for normal connections.

Once the equipment is physically connected, a software connections is required. Network connections are defined by IP numbers and port numbers. The IP identifies the hardware and the port number identifies the software. You will usually need to need to enter the host IP and the host program port number into the OSC device. The IP number can be found in your network settings panel- it looks something like 128.114.11.139 if it was assigned by a DHCP server. IPs on local LANs may start with 10...or 169... If both client and host are on the same router (first three numbers the same) the form computername.local may be used instead of an IP. If both client and host are programs running on the same machine, the IP is 127.0.0.1 or localhost.

The port number may be defined by the host program, or you may just make one up. Made-up numbers must not conflict with any that are already in use, but you will get a warning if you are unlucky. Private port numbers start with 49152-- any number higher than this will probably be safe.

When you enter this in the OSC device, you will notice that the device also has an IP and port number. These will need to be entered in the host software to send data back. To communicate with the device using Max, you need nothing more complicated than Figure 2.

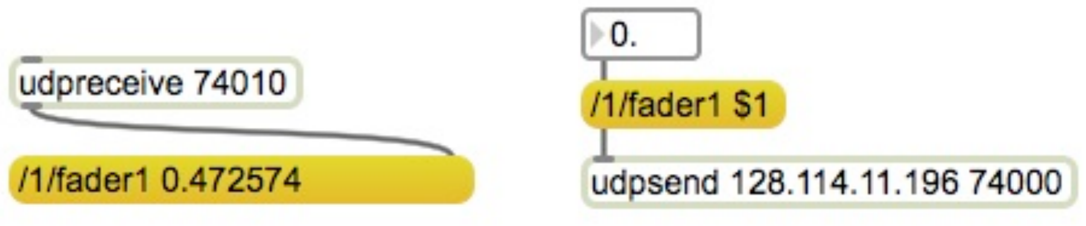


Figure 2.
The udpreceive object takes the host port number as an argument-- this sets the port the patch will listen to. The udpsend object requires the client device IP and port number as arguments.

Some OSC devices and apps provide an easier way to connect. They advertise their availability on a network using a protocol called zeroconfig or "bonjour". This is the system computers use to find printers and so forth. To make this work we need the OSCbonjour object from Rémy Muller at

http://recherche.ircam.fr/equipes/temps-reel/movement/muller/index.php?entry=entry060616-173626

OSCbonjour is illustrated in figure 3.

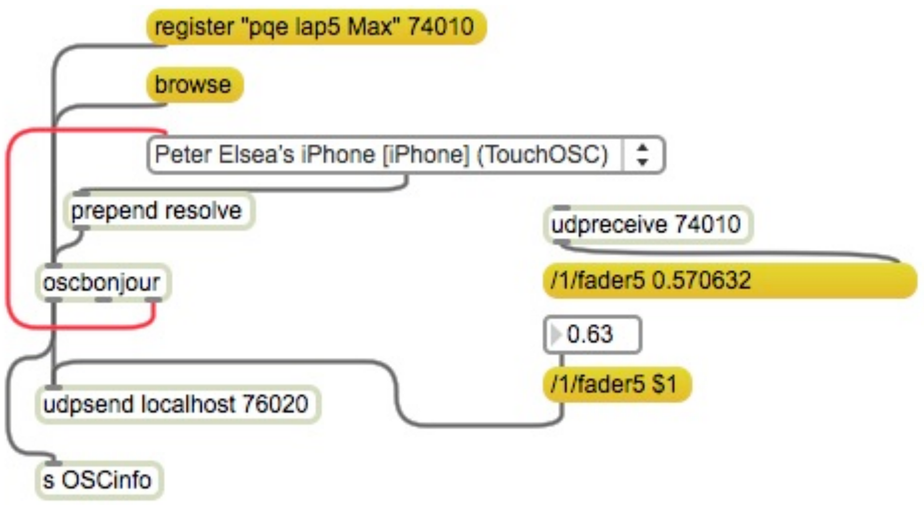


Figure 3. Using OSCbonjour

The OSCbonjour object responds to three messages, which should be performed in this order:

Register followed by your made-up identifier and a port number will advertise that your patch is listening to the port. Once you do this, a bonjour enabled client will list this identifier as a potential host. In TouchOSC you select it on the network setup page.

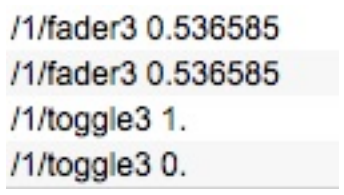
Browse will produce a list of clients that provide a service called _osc._udp. A list of identifiers will come from the right outlet formatted to fill a umenu object.

Resolve with a client identifier will produce host and port messages from the left outlet. These messages will set a udpsend object to send to the client. (The udpsend object must have dummy host and port number to avoid Max errors.)

With figure 3 as part of your patch, setting up the OSC link is as simple as three clicks in max and one in the client. You may need to repeat the resolve message if the client app is shut down and restarted.

OSC Message Formats

Most of the details of OSC communications, such as packet structure, time tags, and data types will be hidden in the software. We are usually presented with an alphanumeric string that is mostly address information followed by one or more values:



```
/1/fader3 0.536585
/1/fader3 0.536585
/1/toggle3 1.
/1/toggle3 0.
```

Figure 3. Typical OSC messages.

The address information is formatted like a URL, with slashes separating the parts. The address identifies both the source and destination of the data. In figure 3 we have values for toggle3 and fader3 of page 1 of the client program. These can be parsed with a route[2] object:

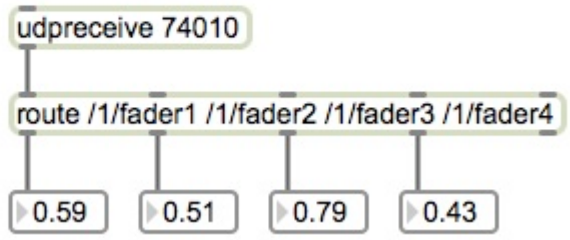


Figure 4.

Route works well as long as the addresses are fairly concise, but some programs are very verbose:

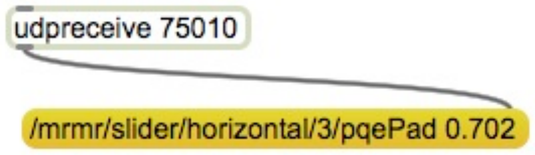


Figure 5.

Route can resolve the entire address, but it makes a very large patch. The CNMAT website[3] features some objects designed to simplify OSC parsing in Max, especially osc-route. This object will break addresses on the slash, and includes wildcard and other pattern matching.

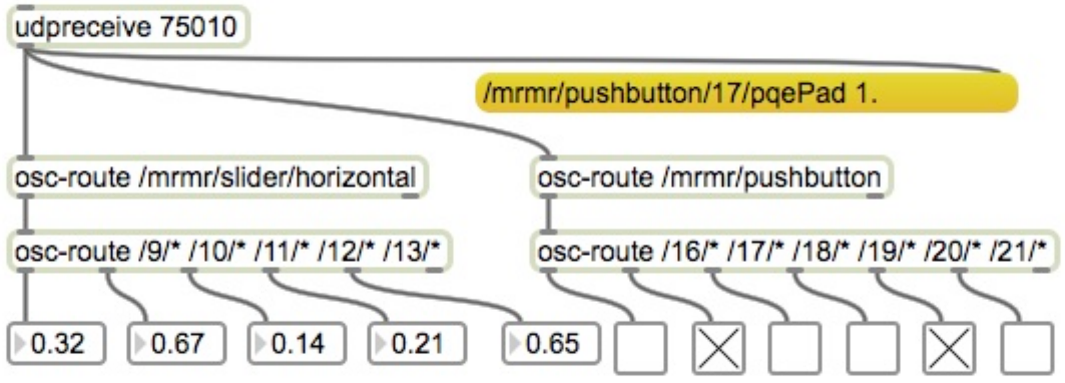

udpreceive 75010
/mrmr/pushbutton/17/pqePad 1.
osc-route /mrmr/slider/horizontal
osc-route /mrmr/pushbutton
osc-route /9/* /10/* /11/* /12/* /13/*
osc-route /16/* /17/* /18/* /19/* /20/* /21/*
0.32
0.67
0.14
0.21
0.65

Figure 6.

The wildcard * matches anything between given characters, so /* will match */pqePad* in figure 6. See the osc-route help file for more options.

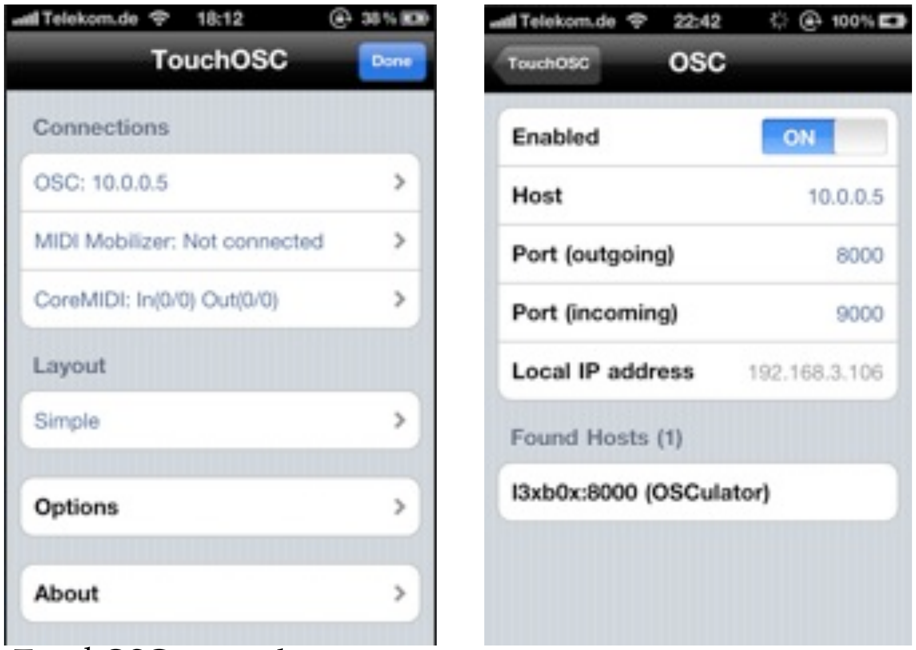
[2] In Max 6 only- previous versions of route did not play well with the slash: you had to write it \/

[3] http://archive.cnmat.berkeley.edu/OpenSoundControl/Max/

Touch OSC

Touch OSC is an iOS and Android app for touchscreen devices. It presents a page of controls that work as sliders and push buttons. These come in an impressive variety, as shown on their web site:

Figure 7. TouchOSC panels in iPad and iPhone.



Telekom.de 18:12 38 %

TouchOSC Done

Connections

OSC: 10.0.0.5

MIDI Mobilizer: Not connected

CoreMIDI: In(0/0) Out(0/0)

Layout

Simple

Options

About

Telekom.de 22:42 100%

TouchOSC OSC

Enabled	ON
Host	10.0.0.5
Port (outgoing)	8000
Port (incoming)	9000
Local IP address	192.168.3.106

Found Hosts (1)

I3xb0x:8000 (OSCulator)

Figure 8. TouchOSC network menu

The setup menu is reached from the info icon on a performance screen. The OSC option brings up the network settings page. The fields need to be filled with:
Host: The IP of the machine running Max
Port (outgoing): The port number in udpreceive.
Port (incoming): The port number in udpsend.

Local IP address must be copied to udpsend.

If you use OSCbonjour for the connection, your identifier should be listed in the Found Hosts field. Selecting this will set up a connection. (Note- bonjour uses different port numbers than those listed in the network page. It is possible to set up two connections by using both bonjour and udpsend arguments, but not a good idea.)

TouchOSC controls and messages

TouchOSC has many types of control. The best way to learn the set is to work with the initial layouts, starting with "simple". Set up a patch like figure 9 with udpreceive connected to a message:

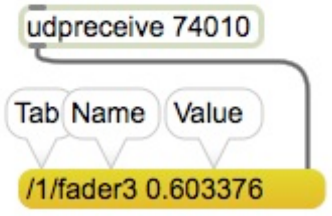


Figure 9.

There will be at least three parts to the message: the tab number, the control name and the value. The default control names are made up of the type with an appended number, but other names can be defined on custom pages. The names of simple devices can be detected with route or osc-route. The osc-route method does make it simple to copy a basic structure for one tab to another. In the right side of figure 10, the faders of tab 2 can be addressed by changing one digit. In fact, you can change the arguments in osc-route with a set message, so you could make a patch that would respond to similar controls on different pages.

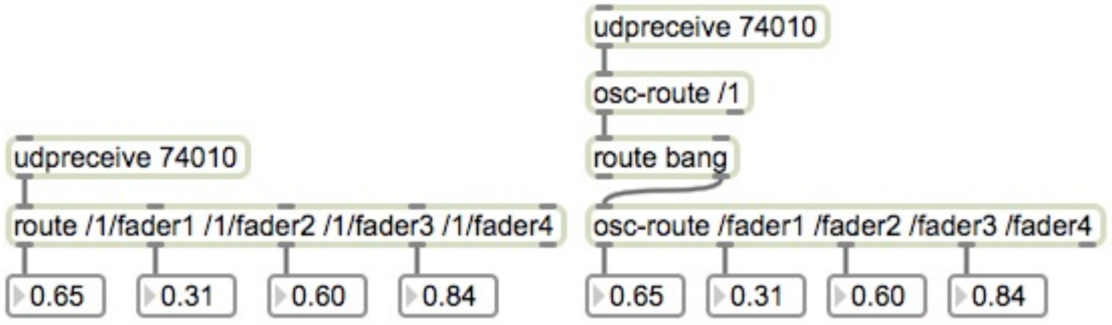

udpreceive 74010
route /1/fader1 /1/fader2 /1/fader3 /1/fader4
0.65
0.31
0.60
0.84
udpreceive 74010
osc-route /1
route bang
osc-route /fader1 /fader2 /fader3 /fader4
0.65
0.31
0.60
0.84

Figure 10.

The tab number is sent whenever it is changed. When osc-route receives an address match with no following value, it bangs, just like route does. Osc-route (and several other objects) post an error message when they receive a bang, so we use *route bang* after the *osc-route /1* in figure 10 and tap off the right outlet. Figure 11 will respond to the current tab.

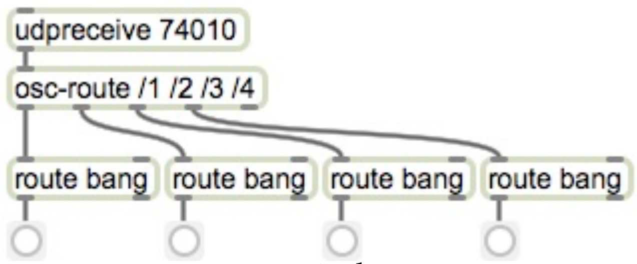


Figure 11. Detecting tab changes.

TouchOSC has several varieties of multi-part control, which provide groups of buttons or faders. The simplest is XY, which outputs two values which must be unpacked:

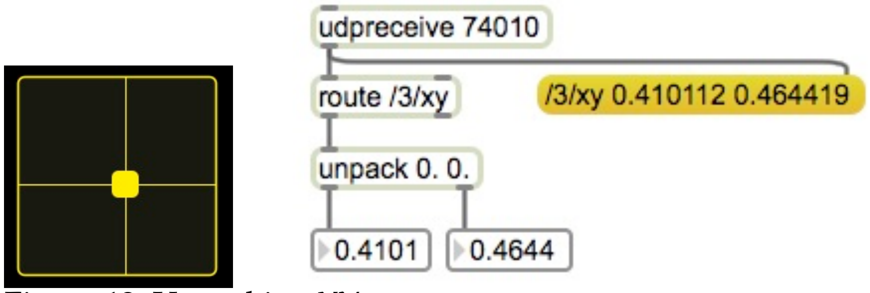


Figure 12. Unpacking XY.

The multifader object is a bit harder to decode, as it includes an extra address item to specify the element moved. A pair of multifaders are found on the fourth page of the mix16 layout. The messages are formatted */4/multifader1/element value*. It would be best if we could get a simple list of element and value to control a multislider object. Unfortunately, osc-route's regular expression tools only extend to recognizing addresses, not modifying them, so we need something more powerful. The regexp object fills the bill.

Regular expressions are specialized scripts that allow loose fitting matches of strings. They are the basis of many search systems and programming systems such as PERL. The most common use of regular expression is the asterisk wildcard used in figure 6, as well as many file retrieval systems. The use of regexp is too complex to completely explain here (there are many on-line tutorials) but we can use one of its most powerful features easily. If we give regexp a symbol as an argument, incoming messages will be tested to see if they contain that symbol[4]. (So-called meta-characters can be included in the symbol to provide complex matches, but we don't need these here.) If we add the attribute @substitute and another symbol, every occurrence of the first symbol will be replaced with the second one. This will be sent from the left outlet. So regexp with the arguments / *@substitute " "* will replace any slash with a space.

[4] Remember, in Max a symbol means any combination of letters, or letters and numerals, even a combination of words if they are enclosed in double quotes.

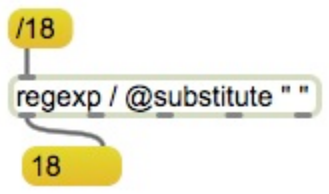


Figure 13.

Figure 14 puts this to work to use a multifader on touchOSC to control a multislider in Max.

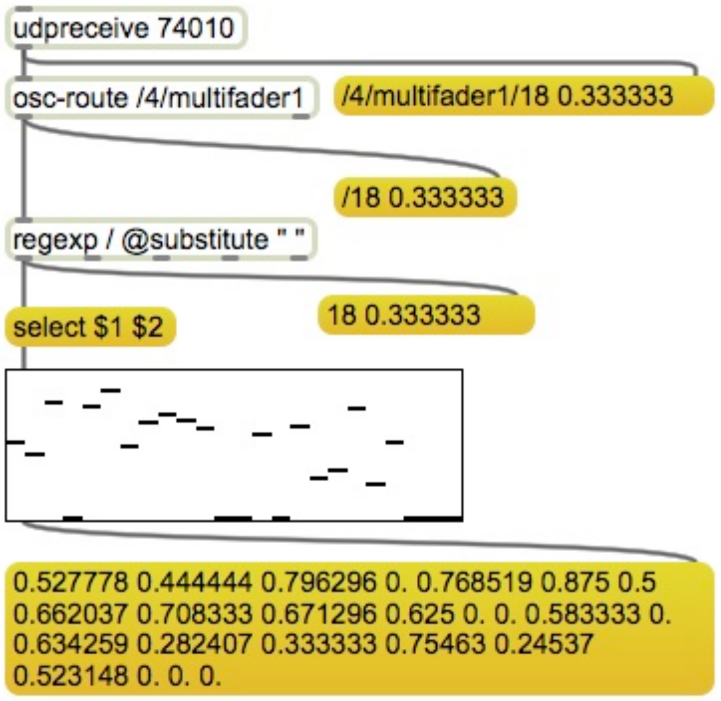


Figure 14.

Multitoggle is a bit more complex, as it uses two address fields to specify row and column. This is no challenge to regexp, as our set of arguments will deal with all slashes, leaving three element lists. Figure 15 shows how to manipulate these lists to manage a matrix control. With regexp the message */2/5 1.* becomes a simple list. Matrix control counts rows and columns from 0, but the touchOSC multi toggle counts from 1. Thus an lsub[5] to subtract 1 from the row and column values in the list.

[5] This is an Lobject, which could be replaced by unpack, subtract and pack.

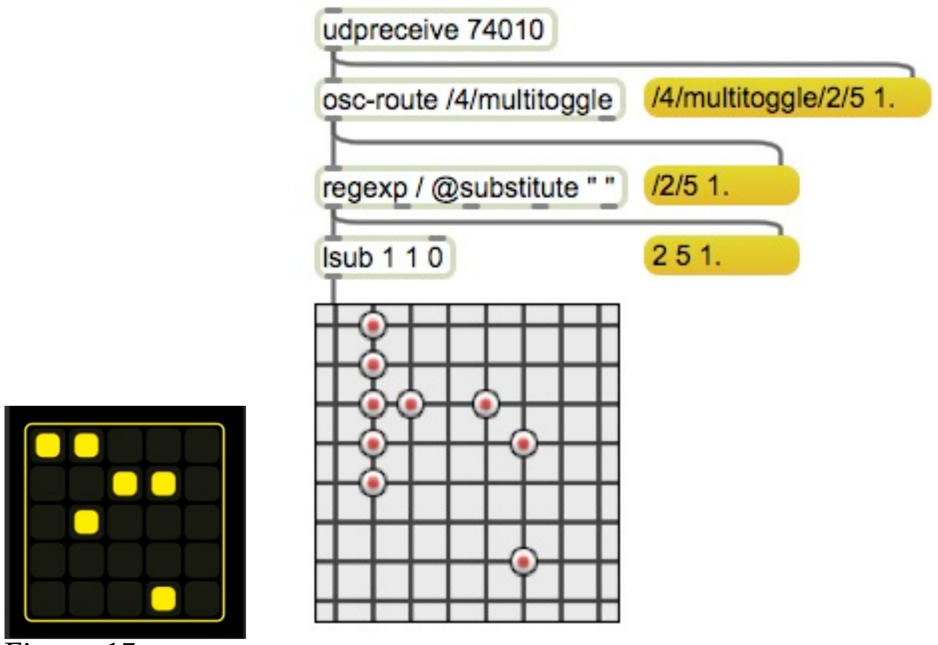


Figure 15.

The last exercise is to implement the "keys" layout, which provides a single octave of piano style keyboard. The control names are pushN where N is the number of the key. Figure 16 shows how to play MIDI notes with this. The match part of the regexp is expanded to */push*, still substituting a space. This will leave a list of key number and 1 for on or 0 for off. With some slight modifications, these can be sent to noteout.

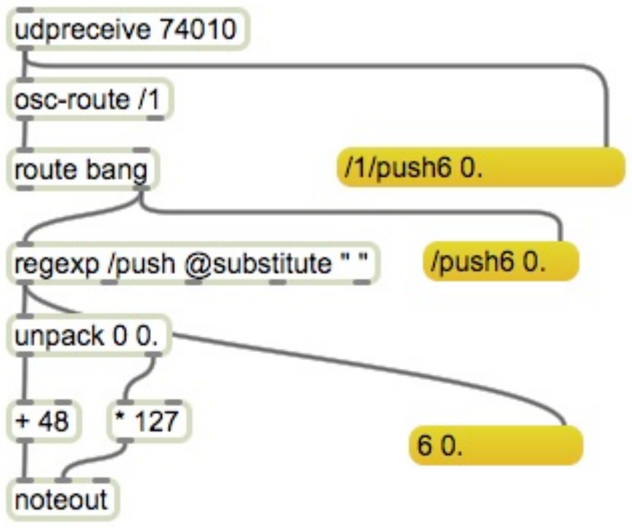


Figure 16.

Other Clients

These techniques should work with any OSC client with minor changes to fit specific commands.