# Open Sound Control

**Brandi Rose**
MSc Student
rose.brandi@gmail.com

**Thomas Haighton**
MSc Student
_@thomashaighton.com

**Danyi Liu**
MSc Student
swell.danyi@gmail.com

**ABSTRACT**
In this document, we describe Open Sound Control (OSC), a protocol used to communicate between computers, sound synthesizers, and other multimedia devices. We will provide a concise overview of the history behind the development of this technology, as well as a brief technical overview. We will also reflect on the strengths and weaknesses of the protocol and describe how applications are currently using Open Sound Control. Finally, we will provide a step-by-step instruction guide to get started with OSC.

## 1. PURPOSE, CONTEXT AND HISTORY

Open Sound Control is a protocol that was developed in 1997 to meet the needs of musicians that wanted more control over sound parameters than the current standard, Musical Instrument Digital Interface (MIDI), could provide. In order to understand how Open Sound Control came to be, let's first look at major milestones in the development of digital instrument interfaces:

- Late 1970's: An increase in the use of electronic musical devices due to affordability [9].
- 1979: The first U.S. recorded digital album is released [2].
- 1981: MIDI is released as a standard peer-to-peer serial port connection between equipment and computers [10]. It uses predefined messages [13] with channel, note, and controller data [8].
- 1994: Zeta Instrument Processor Interface (ZIPI) is developed to address the limitations of MIDI. Instead of peer-to-peer connection, it runs over Ethernet [26]. It attempts to be the new standard, but its unusual addressing scheme was complex and it was never fully adopted [26].
- 1996: Synthesis toolKit Instrument Network Interface (SKINI) is created to extend MIDI features, and allows for messages to be sent as a single line of text [18].
- 1997: Open Sound Control is developed, in an effort to send interactive music over a network protocol (Ethernet) and wanting more control over sound parameters. MIDI was not adequate to represent, organize or name the parameters of their additive synthesizers [23].
- 1998: OSCKit is released as an API developer kit for adding OSC to a real-time system using C library [5].
- 2002: OSC 1.0 is released, as a refined protocol, along with specifications [17].

Open Sound Control's initial purpose was to enable computers, sound synthesizers, and other multimedia devices to communicate with each other [24]. It is a network protocol that is 'transport-independent' and carries messages in binary format [23]. This protocol provides real-time control of sound and other media processing, and has been used in web interactivity tools, software synthesizers, a large variety programming languages, and hardware devices for sensor measurement [16]. In this manner, OSC acts as an enabling middleware between devices to control parameters such as video, audio, pitch levels, color and volume.

## 2. OPERATING PRINCIPLES

OSC supports a client/server architecture, in which a client sends OSC data to a server. In this section we will describe the architectural overview.

OSC requires an IP address and host number for connection, especially to open a remote control from one device to another over a local network. Each service can only listen or send data through its assigned port [13]. For example, if you are using an application on a smartphone to control another application on a computer, you need to set up the corresponding IP address and port number on the smartphone to decide which destination the data will go to, as well as define the same port number on the computer.

### 2.1 OSC packet

The unit of transmission of OSC is an OSC Packet. This can be naturally represented by a network protocol such as User Datagram Protocol (UDP), or stream-based protocol such as Transmission Control Protocol (TCP). The OSC packet must be either an OSC Message or an OSC Bundle [17].

### 2.2 OSC message

The basic unit of OSC data is a message consisting of an address pattern, a type tag string, and arguments.

### 2.3 OSC address pattern

An OSC address pattern is an OSC-string beginning with the forward slash (/), as it is the full path from the root of the address space tree to a particular node, with a slash-delimited format like a Uniform Resource Locator (URL) or file system pathname. For example, the address /voices/3/freq refers to a node named 'freq' that is a child of a node named '3' that is a child of a top-level node named 'voices' [24]. Every OSC server has a set of OSC Methods that can be invoked based on the message's OSC address pattern.

### 2.4 OSC type tag string and arguments

A type tag string is an OSC-string beginning with a comma (,), followed by a sequence of characters corresponding exactly to the sequence of OSC Arguments in the given message [5]. Each character after the comma is called an OSC Type Tag and represents the type of the corresponding OSC Argument [5]. The different data types OSC Arguments support are int32, float32, ASCII strings, and blobs [15]. Other types of data some OSC implementations support are 64-bit numbers, RGBA color, 'True', and 'False' [15].

### 2.5 OSC bundle and time tag

A bundle is a sequence of messages and/or bundles, which consists of the OSC-string '#bundle', and an OSC Time Tag, followed by zero or more OSC Bundle Elements. Time tag is used to let the server get access to the representation of the absolute time. If a received OSC packet contains only a single message, the OSC server should unpack it and invoke the corresponding OSC method. Otherwise the OSC Time Tag determines when the OSC methods should be invoked immediately (see Figure 1) [5]. If the time tag represents a time in the future, the server should store the bundle until the specified time arrives. If the time tag is equal to or before the current time, the server should unpack it and invoke the method immediately [5].

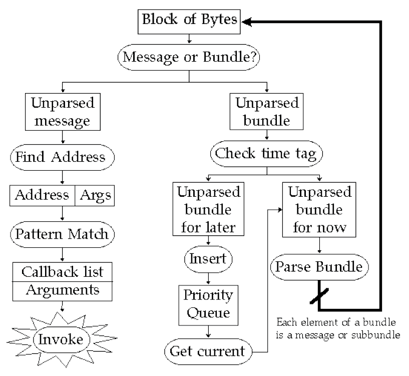

**Figure 1. Life of an OSC packet [5]**

## 3. STRENGTHS AND WEAKNESSES

In this section we will discuss the strengths and weaknesses of OSC compared to MIDI. The MIDI protocol was developed by a group of American and Japanese synthesizer manufacturers, who wanted to create a standard protocol to transfer musical data between instruments. This standardization was accomplished by having pre-defined messages, a standard transfer protocol and a standard file format. Any MIDI capable device can connect to any other MIDI device and communicate. There are thousands of software and hardware products that support MIDI, compared to the dozens that support OSC [7].

In contrast, OSC is an open source network protocol with a dynamic, open ended, URL style-naming scheme, which allows the user to determine the address space [16]. In section 2, we have already mentioned the specifics of OSC, which are a large part of its uniqueness and strengths. Open Sound Control's main strengths are the naming scheme, numeric and symbolic argument messages, pattern matching language, high-resolution time tags, bundles of messages, and a query system to dynamically find the capabilities of an OSC server and get documentation [14]. OSC has a lot of advantages over MIDI (see Table 1).

| | OSC | MIDI |
|---|---|---|
| Examples of Messages | `/wii/ir/x 0.1503`<br>`/stylus/pressure 0.814`<br>`/camera/look-at 5. 12. 17.`<br>`/play-note 15 0.9` | `144 60 64 (MIDI Note-on)`<br>`128 60 64 (MIDI Note-off)` |
| Message types | User-defined<br>Human-readable | Pre-determined<br>Byte-encoded |
| Atomic Update of Multiple Parameters | ✓ via Bundles | ☹ |
| Time-tagging | ✓ via Bundles | ☹ |
| Hardware Transport Independent | ✓ | ☹ |
| Number of channels | Unlimited | 16 |
| Data formats | Integer, Double Precision Floating Point, Strings, and more | 1-byte integers 0-255 |
| Message Structure | User-defined | Pre-determined |
| Microcontroller-friendly | ✓ | ✓ |
| State-machine independent | ✓ * (Unless user-imposed) | ☹ (e.g. "The Note-off problem") |
| Application Areas | Music,, Video, Robotics and more | Music |
| Clock-sync accuracy, theoretical limit | picosecond (via NTP / IEEE 1588 Sync) | 20833 microseconds |
| Data Rate | Gigabit Speed (> 800M bits / sec) | 31,250 bits / second |

**Table 1. Table comparison of OSC and MIDI [3].**

It's biggest strength, which allows each user to design an address space instead of selecting a predetermined set of parameter names, is also its weakness because it is easily adaptable to situations never envisioned by the designers [23]. The MIDI protocol uses a compact binary message format with a limited number of fields [22]. Because each user can name their own address spaces, these fields can easily be identified by name. The lack of standardization of OSC is one of the reasons it has never replaced MIDI.

OSC also has the ability to send a continuous stream of 32bit floating point numbers, in contrast to MIDI who mainly sends 7 bit integer values. This makes OSC a much better alternative when dealing with electronic sensor-based instruments, which can register minute differences in changes. There is however a way around this limitation of MIDI, by defining System Exclusive messages that use other data formats such as strings and floating points [3].

Additionally, OSC includes a high-precision timestamp with picosecond-resolution that allows OSC messages to be scheduled, recorded and reproduced with minimal jitter. The MIDI beat-clock is a low-resolution clock having precision on the order of several milliseconds at best [22]. The MIDI protocol and transport do not have time-tagging because they are intended for immediate delivery.

Another weakness of OSC is that it has a certain latency and interference, which can cause jitter. Using the time tags of OSC bundles, events can be scheduled with fixed delays that account for the network transport delay in communication between devices. Jitter is randomness in time and may be found in the transport delay, or in the clock synchronization error. If the clock synchronization error is smaller than the transport jitter, which is often the case, and the data stream uses timestamps, then it is possible to use forward synchronization to remove jitter from a control signal [19].

## 4. TYPICAL APPLICATIONS

OSC is the successor of the MIDI protocol, so typical applications of OSC are originally in the same area as MIDI applications. Open Sound Control's typical application areas are Sensor/Gesture-based electronic musical instruments, mapping nonmusical data to sound, multiple-user shared musical control, web interface to sound synthesis, networked Local Area Network (LAN) musical performance, Wide Area Network (WAN) performance, and virtual reality [24]. We will describe two typical applications of how OSC is being used in electronic music production and performance.

### Tablet as an interface

TouchOSC is a tablet application that musicians use as a wireless device to control musical software on a computer. Developed by hexler.net, the app runs on Apple and Google devices, and is able to send and receive OSC and MIDI messages over Wi-Fi and control CoreMIDI compatible software, hardware and mobile apps [4].

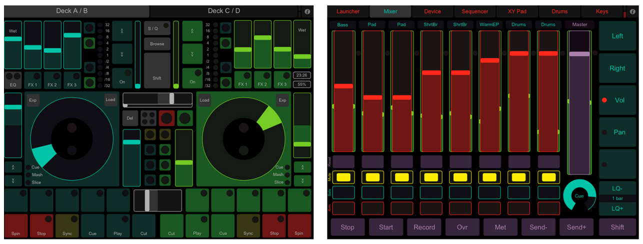

**Figure 2. TouchOSC interfaces**

The interface provides visual representations of buttons, faders, rotary dials and x-y controllers arranged in conventional ways (see Figure 2). The user can also personalize the layout by using the TouchOSC Editor software. Each interface element has a specific OSC address: /tab_number/interface_element/value and has to be mapped by the user to a parameter in the host application. TouchOSC comes with pre-mapped layouts, which are used to control Digital Audio Workstations (DAW) like Ableton Live and Apple Logic. Many apps similar to this exist, but they all work the same way, by using the OSC protocol to enable the tablet and computer to communicate with each other.

### Sensor/Gesture-based electronic music

Because of MIDI's low resolution, 128 integer values and thus only 128 steps to control a musical parameter, electronic musicians and composers use OSC to overcome this limitation. Having a large resolution means the transitions between values are smoother, and therefore an electronic musician/composer can make more subtle and smoother changes, which gives them more expressive control. Electronic musicians have been experimenting a lot these past decades on how to get to a level of control that mimics the intimacy a professional player has when playing a traditional musical instrument [21]. Most endeavors involve creating personalized instruments that implements sensors, which can capture bodily movements. In addition, each physical gesture has to be mapped to a parameter of the sound.

One such endeavor is the Mi.Mu glove, used and co-developed by the artist Imogen Heap [6]. The Mi.Mu features motion tracking via an on-board Inertial Measurement Unit (IMU) to enable real-time gesture detection. The haptic motors provide the user with tactile feedback without needing to see a computer screen, and the flex/bend sensors over the knuckles track the orientation of your hand. The Mi.Mu also has a micro controller called the x-OSC, which takes data from all the sensors in the glove and sends them as OSC data over Wi-Fi to an OSC server.

## 5. SURPRISING APPLICATIONS

Nowadays, artists are using OSC in surprising ways that have nothing to do with controlling sound. We will go over two, out of the many existing applications, using OSC in a manner other than its intended use of controlling sound.

### Play Array

Recently, a public interactive art piece called Play Array was installed in the New York University Center for Urban Science and Progress Pop Up. This interactive installation used OSC to have two devices communicate with each other [20]. The piece was a large pixel grid of virtual pong, which allowed users with smartphones to play the game. The pixel grid consisted of large LED circles that were controlled by an Arduino Pro Mini, by which it received its instructions over Ethernet via communication from the data from the smartphone [20]. Open Sound Control was used to enable the communication and send the data from the phone to the Arduino that controlled the LEDs [20].

### Silhouette Theater

Another surprising use of OSC was implemented in a project called Silhouette Theater by three Media Technology students at Leiden University. They developed a modern version of the Chinese shadow puppetry Pi Ying Xi, an ancient method of storytelling [12]. In this project, OSC was used in two ways; 1) to send data from an Xbox Kinect to Max/MSP to control live music and 2) to send the data to Animata to control the animated puppet [12]. The user stands in front of the Kinect, which tracks the user's body movements, and thus the movements of the animated puppet. These movements control the sound and the animation projected on a screen. This application is an example of using the OSC protocol to connect with other applications in real time.

## 6. GETTING STARTED

### 6.1. Requirements

The following guide uses Processing, an open source programming language, to describe how to send and receive data via Open Sound Control protocol on one computer. Before getting started you need to download and install Processing 2.0 or 3.0 from https://processing.org/download.

### 6.2. Step-by-step instruction

#### *6.2.1. Setting up an OSC connection*

After opening a new sketch in Processing, install oscP5 library by going to Sketch -> Import Library -> Add Library, then add the following code to the top of your sketch to import the library [1]:

```
import oscP5.*;
import netP5.*;
```

Next, we create the object OscP5 for sending OSC messages, and the object NetAddress for storing a network address where we are going to send the OSC messages to:

```
oscP5 oscP5;
NetAddress myRemoteLocation;
```

In order to set up an OSC connection, a port number for listening or sending data needs to be assigned. We will send and receive data on the same computer, thus we initialize the oscP5 object connecting to port 7200, add IP address (in this case 127.0.0.1 is the local loopback address) and the same port number to the myRemoteLocation object. If you want to open a remote connection from computer A to computer B, then you need to proceed with a local connection and find the IP address of computer B:

```
void setup() {
   //listening for incoming messages at port 7200
  oscP5 = new OscP5(this,7200);
   //sending OSC message to the device at port 7200
  myRemoteLocation = new NetAddress("127.0.0.1", 7200);
}
```

#### *6.2.2 Sending and receiving OSC messages*

In this example, we are using mousePressed() function to allow the application to send OSC messages on every mouse click. Define address with a forward slash (/), which helps you route the data on the receiving end. Use myMessage.add(); to add different types of data to send at the same time, and send it to the location we specified in myRemoteLocation (section 6.2.1):

```
void mousePressed() {
 // create an OSC message with address pattern
OscMessage myMessage = new OscMessage("/test");
 // add an int to the osc message
 myMessage.add(123);
 // add a float to the osc message
 myMessage.add(12.34);
 // add a string to the osc message
 myMessage.add("some text");
 oscP5.send(myMessage, myRemoteLocation);
}
```

The received OSC messages are passed to the oscEvent(OscMessage OscMessage) function in Processing. Add the following code to your sketch to receive and parse messages:

```
void oscEvent(OscMessage theOscMessage) {
// check theOscMessage the address pattern
  if (theOscMessage.checkAddrPattern("/test") == true) {
// get the first osc argument
    int firstValue = theOscMessage.get(0).intValue();
// get the second osc argument
    float secondValue = theOscMessage.get(1).floatValue();
// get the third osc argument
    String thirdValue = theOscMessage.get(2).stringValue();
   println("Received values: " + firstValue + " "  +
secondValue + " " + thirdValue);
   return;
  }
}
```

Firstly, check whether the address pattern matches since there might be a lot of messages coming in with different addresses. Secondly, define the corresponding data type of each osc argument. If the incoming argument is an integer, but the data type is defined as a float on the receiving end, then the argument will fail to invoke.

#### *6.2.3. Test the OSC connection*

The application can be tested within println();. Press run, then you will see the OSC messages printed out in the log window if it succeeds.

## 7. FINAL THOUGHTS

Currently there is no universal protocol for music equipment, mainly because there are too many different synthesis methods, programming systems, levels of user control, and forms of sound manipulation [11]. Still, OSC continues to be a lasting web technology that is enabling musicians and artists to do more than just control sound. Open Sound Control is great for interactive and collaborative work, networked art and multiplayer games, and using gestures or sensory input to control parameters.

When having to choose between MIDI and OSC, it is important to think about who is going to use it and for what purpose. If a musical hardware device has to interface with other commercial software and hardware products, it is advisable to implement MIDI. It will be closer to a plug-and-play device, which makes it easier to use for a regular consumer. If a device is created for individual use, it is easier to implement OSC, because of the ease of implementation in different programming languages and support through the open source community.

## REFERENCES

1. Codasign. http://learning.codasign.com/index.php?title=Sending_and_Receiving_OSC_Data_Using_Processing
2. Digital Recording. http://en.wikipedia.org/wiki/Digital_recording
3. Freed, A., Schmeder, A. and Zbyszynski, M., Open Sound Control -- A flexible protocol for sensor networking, Center for New Music & Audio Technologies. http://opensoundcontrol.org/files/OSC-Demo.pdf
4. Hexler.net, TouchOSC. http://hexler.net/software/touchosc
5. Implementation and Performance Issues with OpenSound Control, The OpenSound Control Kit. http://archive.cnmat.berkeley.edu/OpenSoundControl/Kit/ICMC98-presentation/OSC-Kit-ICMC98.html
6. Mi.Mu Gloves. http://mimugloves.com
7. MIDI Manufacturers Association Incorporated. http://www.midi.org/aboutmidi/midi-osc.php
8. MIDI. http://en.flossmanuals.net/pure-data/midi/using-midi/
9. MIDI. http://en.wikipedia.org/wiki/MIDI
10. MIDI. http://en.wikipedia.org/wiki/MIDI#The_development_of_MIDI
11. Musicradar.com, 30 years of MIDI: a brief history (2012). http://www.musicradar.com/news/tech/30-years-of-midi-a-brief-history-568009
12. NMNT 2015. https://sites.google.com/site/newmedianewtechnology2015/portfolios/danyi/lab1
13. Open Sound Control. http://en.flossmanuals.net/pure-data/ch065_osc/
14. OpenSound Control Home Page. http://archive.cnmat.berkeley.edu/OpenSoundControl/
15. Opensoundcontrol.org. http://opensoundcontrol.org/implementation/osc-net-v1-2
16. Opensoundcontrol.org. http://opensoundcontrol.org/introduction-osc
17. Opensoundcontrol.org. http://opensoundcontrol.org/spec-1_0
18. Princeton University Computer Science Department, Synthesis toolKit Instrument Network Interface (SKINI). https://ccrma.stanford.edu/software/stk/skini.html
19. Schmeder, A., Freed, A. and Wessel, D., Best Practices for Open Sound Control. In *Linux Audio Conference*, 2010, 3. http://opensoundcontrol.org/publication/best-practices-open-sound-control
20. Urban Matter Inc. http://urbanmatterinc.com/play-array-urban-pong-game/
21. Wessel, D. and Wright, M., *Problems and Prospects for Intimate Musical Control of Computers*. Center for New Music and Audio Technologies, Berkeley, CA, USA, 2001.
22. What is the difference between OSC and MIDI? http://opensoundcontrol.org/what-difference-between-osc-and-midi
23. Wright, M. Open Sound Control: an enabling technology for musical networking in Organised Sound 10(3), Cambridge University Press (2005), 193-200.
24. Wright, M., Freed, A., Momeni A.: “OpenSound Control: State of the Art 2003”. Proceedings of the 3rd Conference on New Instruments for Musical Expression (NIME 03), Montreal, Canada, (2003), 153.
25. Wright, M., OpenSoundControl Application Areas, 2004. http://archive.cnmat.berkeley.edu/OpenSoundControl/application-areas.html
26. Zeta Instrument Processor Interface. http://en.wikipedia.org/wiki/Zeta_Instrument_Processor_Interface