Manual

Version 2.2.1

Asset Store Description

An Open Sound Control (OSC) implementation tailored for Unity and designed for those who love the flexibility of scripting. OSC is a protocol for communicating between applications and devices easily using URL-style messages with mixed argument types. It is widely used in "creative coding", music and VJ contexts, but also has potential as a general purpose networking tool.

Requires basic experience with C# scripting.

Features

- ZERO heap garbage in the update loop! (*1)
- All OSC argument types (*2)
- Bundles with timetags
- Full two-way OSC address pattern matching
- Mapping of OSC addresses to methods
- Common Unity types as blobs
- Monitoring of incoming and outgoing messages
- Monitoring of remote connection status
- Optional filtering of message duplicates
- Optional auto bundling of messages
- Optional buffer safe splitting of large bundles

Supports

- UDP IPv4 Unicast, Broadcast and Multicast
- Scripting Runtime Version 4.x (not 3.5)
- API Compatibility Level 2.0 and 4.x
- MacOS, iOS and Windows (*3)

Includes

- Manual
- Reference
- Examples
- Full source code
- Runtime UI prefabs

Tested with

OpenFrameworks, Processing, Max/MSP, VVVV, TouchOSC, Lemur, Iannix and Vezer.

*1) If used as advised and your strings and blob lengths don't change.
*2) Except arrays.
*3) Not officially supported on other platforms, but it may work.

Underlying Concepts

OSC simpl transmits messages targeting IPv4 addresses over unicast, broadcast and multicast UDP.
If those words sound familiar to you then skip this page.

OSC

Open Sound Control (OSC) is network protocol initiated in 1997 and developed at Center for New Music and Audio Technologies (CNMAT). All about OSC at http://opensoundcontrol.org/

UDP

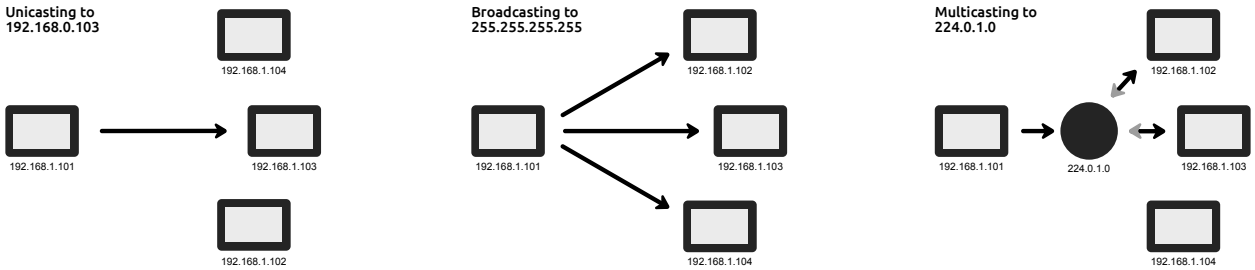
The User Datagram Protocol (UDP) is a network protocol that offers very fasts but unreliable transmission. Contrary to the The File Transfer Protocol (FTP), UDP has no native mechanism for checking if messages reach their destination. No connection is established. The messages are simply send to a network destination. All about UDP at https://en.wikipedia.org/wiki/User_Datagram_Protocol

IPv4

The Internet Protocol version 4 (IPv4) is a version of the Internet Protocol (IP). For the purpose of *OSC simpl*, all we need to know is that IPv4 defines the format of the IP addresses we use to target devices. The format is XXX.XXX.XXX.XXX, where XXX is an integer between 0 to 255. For example; 192.168.1.101.

Unicast, Broadcast and Multicast

IPv4 offers three modes of transmission are available; unicast, broadcast and multicast. The mode is defined by the address number. Multicast ranges from 224.0.0.0 to 239.255.255.255, broadcast is always 255.255.255.25 and unicast occupies the remaining addresses. For all modes, the sender needs to provide a target port number that the target device can listen on.



Unicast

Unicast transmission is used for targeting a singe device. A sender needs to know the IP address of the target device and a port number for applications to listen on. The address 127.0.0.1 is called the loopback address, and is used for sending to other applications on the same device. Unicasting is the fastest and most reliable mode of transmission.

For applications where low latency is critical, use unicast. Unless you are sending to many devices, unicast offers better performance than broadcast and multicast.

Broadcast

Broadcast transmission is used for targeting all devices on the local network. A sender must send to the global address 255.255.255.255. Broadcasting is the slowest method of sending, but it is useful because the sender does not need to know any IP addresses.

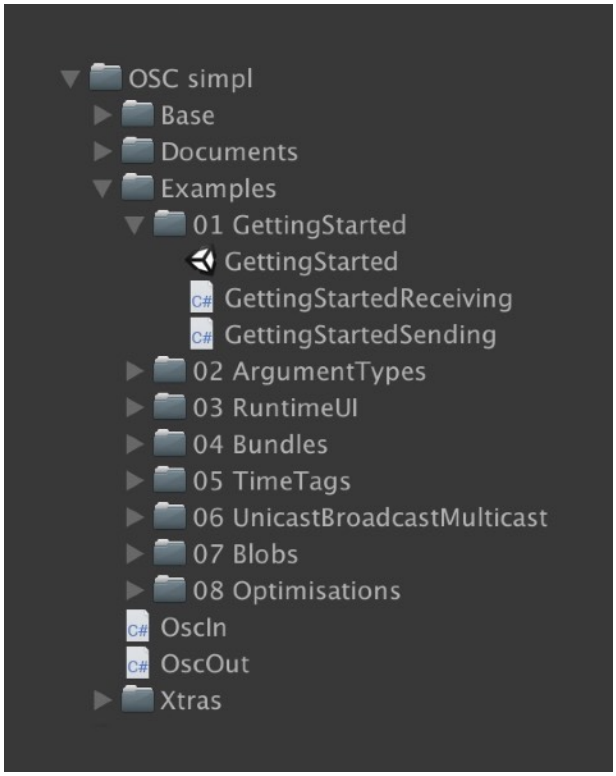
Multicast

Multicast is used for targeting a "multicast group" that applications on other devices may be listening to. A sender must know a valid multicast address, ranging between 224.0.0.0 and 239.255.255.255. A receiver must also know the multicast address. Multicasting is potentially faster than broadcasting, but slower than unicasting.

For multicast to work properly, all routers involved must be "multicast enabled" (most routers are).

Getting started

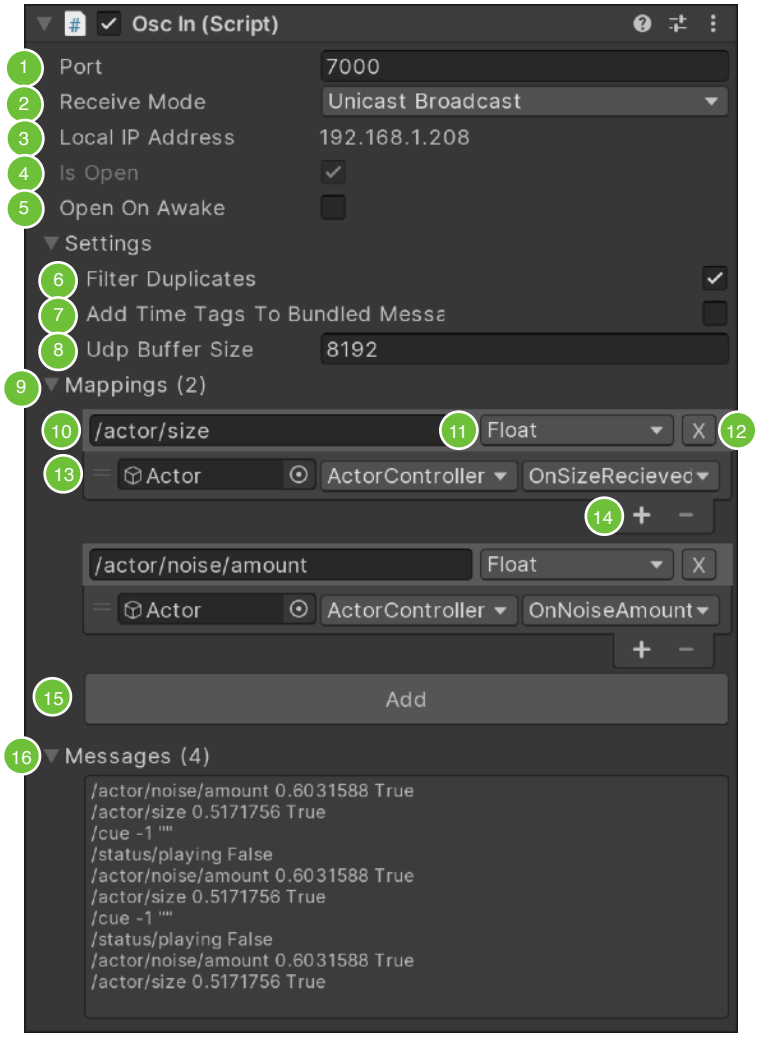
Learn by doing. Dive directly into the examples and get your hands dirty.



Inspector Panel Reference

Below is a description of each field in the inspector panels for the components *OscIn* and *OscOut*. In most cases, the fields are exactly the same as the public properties found in the included OSC simpl/Documents/Reference.pdf.

OscIn



1. Port

The local network port that this component is set to listen on.

2. Receive Mode

The type of transmission the component will listen to. If set to UnicastBroadcastMulticast, and additional Multicast Address field needs to be filled.

3. Local IP Address

The local network IP address that this device will listen on.

4. Is Open

Indicates whether the component is open for incoming messages.

5. Open On Awake

When enabled, the component will automatically open when it gets the Awake call from Unity. Default is false.

6. Filter Duplicates

When enabled, only one message per OSC address will be forwarded every Update call. The last (newest) message received will be used. Default is true.

7. Add Time Tags To Bundles Messages

When enabled, timetags from bundles are added to contained messages as last argument. Incoming bundles are never exposed, so if you want to access a time tag from an incoming bundle then enable this. Default is false.

8. UDP Buffer Size

Set the internal buffer size of the UDP socket.

9. Mappings

A list of mappings that bind messages with specific OSC Addresses and OSC Argument types to Unity methods.

10.

Expected OSC Address.

11.

Expected type of the first OSC Argument.

13.

Taget GameObject, Component and method.

14.

Add and remove methods.

12 & 15.

Remove and add mapping.

16 Messages

Console showing incoming messages.

OscOut

1. Port

The remote network port that this component will send to.

2. Send Mode

The type of transmission the component will send. The mode is derived from the *Target IP Address* field.

3. Target IP Address

The remote network IP address that this application will send to.

4. Is Open

Indicates whether the component is open and ready to send messages.

5. Open On Awake

When enabled, the component will automatically open when it gets the Awake call from Unity. Default is false.

6. Multicast Loopback

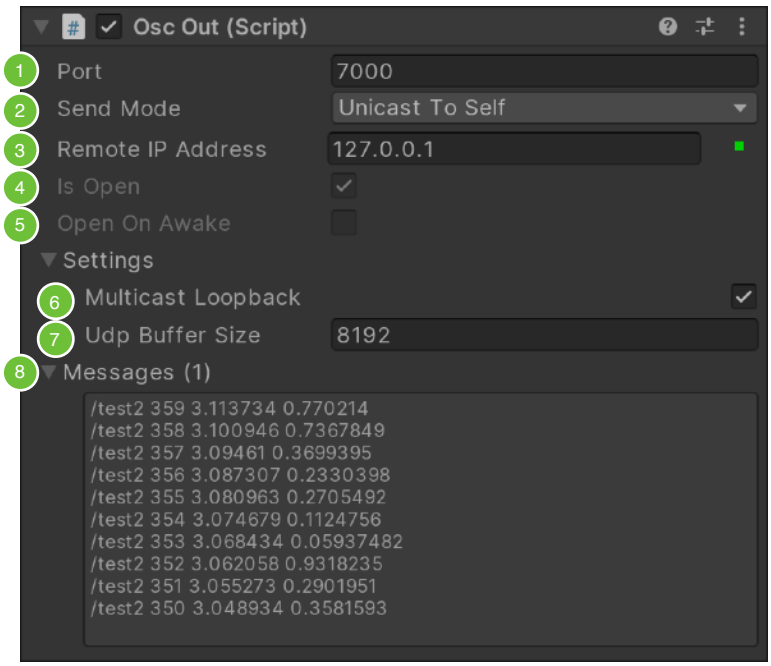
Indicates whether outgoing multicast messages are also delivered to the sending application. Default is true.

7. UDP Buffer Size

Set the internal buffer size of the UDP socket.

8. Messages

Console showing outgoing messages.



Other Topics

Timetags

In *OSC simpl* timetags are represented by the type OscTimeTag. OscTimeTag objects are send implicitly with bundles and explicitly as message arguments.

Incoming bundles are never exposed to the user. Instead, the contained messages are unwrapped automatically and send to mapped methods. If you want to receive timetags from bundles, then enable 'addTimeTagsToBundledMessages' on OscIn and grab the timetag from the last argument of your incoming bundled message.

OscTimeTag supports a precision of 0.0001 milliseconds (one DateTime tick) using the 'time' property, and a precision of about 0.0000002 milliseconds if you manipulate the 'oscNtp' property directly (not recommended unless you understand the protocol). Note that the precision of DateTime.Now is about 1 millisecond.

Address Pattern Matching

OSC simpl supports OSC 1.0 style Address Pattern Matching. In the original 1.0 design, pattern matching is one way: OSC messages hold OSC Address Patterns that target OSC Methods that hold OSC Addresses. In practise, it is also useful for OscMappings to hold patterns, that is why pattern matching in *OSC simpl* is two-way. When conflicting patterns are found (when both a OscMessage and a OscMapping have patterns in the same parts of the address) they are simply ignored.

So what exactly is a OSC Address Pattern? These are examples:

Pattern	**Match**
/*	One address part containing anything.
/*/*	Two address parts containing anything.
/lights/*	/lights/ followed by one address part containing anything.
/{cat,dog}	/cat and /dog
/[ch]at	/cat and /hat
/?at	/cat, /hat /bat and anything else you can replace the first character with.
/synth/[2-4]	/synth/2, /synth/3 and /synth/4

See the OSC 1.0 specification for a full explanation:
http://opensoundcontrol.org/spec-1_0

Supported OSC Type Tags

OSC Type Tag	Meaning	Unity Data Type	Compatibility	Byte Count	Notes
f	32 bit float	float	OSC 1.0, OSC 1.1	4	
d	64 bit float	double	OSC 1.0 nonstandard	8	
i	32 bit integer	int	OSC 1.0, OSC 1.1	4	
h	64 bit integer	long	OSC 1.0 nonstandard	8	
s	ASCII string	string	OSC 1.0, OSC 1.1	Variable	One to four trailing zeros, multiple of four.
S	ASCII string	string	OSC 1.0 nonstandard	Variable	One to four trailing zeros, multiple of four.
c	ASCII char	char	OSC 1.0 nonstandard	4	
T	Boolean True	bool	OSC 1.0 nonstandard, OSC 1.1	0	
F	Boolean False	bool	OSC 1.0 nonstandard, OSC 1.1	0	
r	32 bit rgba color	Color32	OSC 1.0 nonstandard	4	
b	Blob	byte[]	OSC 1.0, OSC 1.1	Variable	Multiple of four.
t	OSC Time Tag	OscTimeTag	OSC 1.0 nonstandard, OSC 1.1	8	
m	MIDI message	OscMidiMessage	OSC 1.0 nonstandard	4	
N	Null		OSC 1.0 nonstandard	0	
I	Impulse		OSC 1.0 nonstandard, OSC 1.1	0	Was called “Infinitum” in OSC 1.0 and was later referred to as “Impulse” in OSC 1.1.

What about OSC Arrays?

The OSC 1.0 specification mentions arrays, signified by OSC Type Tags ‘[‘ and ‘]’. However, the way they are defined, they are actually lists of mixed types, similar to the OSC Arguments themselves. This complicates the way arguments are get and set in *OSC simpl* where System.Object is avoided at any cost (to eliminate garbage generated by “boxing”). OSC Arrays will be supported in the future if an elegant zero garbage solution is found.

Sources

OSC spec 1.0 http://opensoundcontrol.org/spec-1_0
OSC spec 1.1 http://www.nime.org/proceedings/2009/nime2009_116.pdf

Troubleshooting

No messages are received

Step 1: Obtain IP address

How to get the IP address depends on your system.

> [MacOS] Go to *System Preferences –> Network* and select the adapter you want to connect through.
> [Windows] Go to *Settings –> Network & Internet* and select *View your network properties*.
> [iOS] Go to Settings –> Wi-Fi and press the info ion next to the network you wish to connect through.

If you are building an app that is using OSC simpl to your device using, then you can use *OscIn.localIpAddress* to obtain the primary IP address. Beware that on iOS, your mobile internet IP may be returned instead of your Wifi IP. In that case, use OscIn.localIpAddressAlternatives to find the right one.

Step 2: Check connection

Send a ping message between the devices you want to connect.

> [MacOS] Open the Terminal app and write *ping* followed by the target IP address (for example *ping 192.168.1.39*).
> [Windows] Open the Command Prompt and do the same.

No replies? Ensure that the two devices are physically connected to the the same network and they obtain an IP address from the same network. Still no progress? Continue below.

[Windows]
On Windows 10, you must enable “File and printer sharing” for messages to come through. Go to Windows Settings –> Network & Internet –> Network and Sharing Centre –> Change advanced sharing settings –> File and printer sharing. Enable “Turn on file and printer sharing”. Still no pings coming though? Continue below.

[MacOS]
On MacOS, in some cases, ping messages don’t reach Windows machines when the MacOS device are connected though multiple network adapters. If you are trying to connect via ethernet, then try to turn off your wifi and ping again. Still no pings coming though? Continue below.

Step 3: Firewall

Check your network sharing settings.

[MacOS] Go to System Preferences –> Security & Privacy –> Firewall –> Firewall Options. Find your Unity app or your build app and allow incoming and outgoing connections.

[Windows] Go to Windows Settings –> Network & Internet –> Windows Firewall –> Allow an app through the wall. Find your Unity app or your build app and allow incoming and outgoing connections. If OscIn still does not receive anything, then also try go to Windows Settings –> Network & Internet –> Windows Firewall –> Advanced Settings -> Inbounds Rules. Here, you may also have to find your Unity app or your build app and allow incoming connections.

At this point it should be possible to send and recieve pings between the two devices. Now open Unity and try again.

Step 4: Port

Make sure you are sending and receiving on the same port. Also make sure that no other applications are using that port.

Some messages are lost

UDP limitations

OSC simpl relies on UDP, a very fast but unreliable network protocol. UDP does not guarantee that messages reach their destination. If the routers involved in your network are too busy then messages may get lost.

Internet limitations

If you are sending across the internet, then keep the size of your data packets below 512 bytes to increase the chance of survival.

[OscIn] Multiple OscIn objects with same port

OSC simple allows multiple OscIn objects with same port, however only one of them will be receiving at the same time. That's the nature of sockets. Even in cases where multiple applications can access the same port, the data is handed out first-come, first-serve. Only one socket will receive.

[OscIn] Error occurred while receiving message.

If you get this warning in the console, and the next line spells "System.ArgumentException: length", then it is likely that the package size of the message was too big. In the authors tests, the size limit for packages send in broadcast mode is 1472 bytes. This limit may vary depending on the system.

Incoming bundles are not dispatched at timetag time

Timed scheduling of incoming bundled messages is not supported. All messages are dispatched immediately. This is the case for most OSC implementations and complies with a paper published in 2009 describing the forthcoming OSC 1.1.

Incoming messages are always one frame delayed

Script Execution Order

OscIn needs to be executed first to ensure low latency. When you inspect a *OscIn* component in the Unity Editor, OscIn is automatically set to be first in the Script Execution Order. In the strange case that you never inspect an OscIn components, you have to do it manually, just once. Find the setting under Project Settings.

Known issues

MacOS 10.14.2 build interruptions

MacOS builds are interrupted when running in Window mode and hidden behind other windows, even though Application.runInBackground is enabled. This causes regular interruptions in messages send by OSC simpl.
https://forum.unity.com/threads/macos-build-has-update-loop-hiccups-in-window-mode.627781/

Support

If you wish to contact the author for support, then please make sure that problem cannot be answered by this manual, the scripting reference or the provided examples. You can write the author on the related forum thread:

https://forum.unity.com/threads/released-osc-simpl.382244

License

OSC simpl is a Unity Asset Store product created by Danish interaction design consultancy Sixth Sensor. Please read the End User License Agreement on Unity's website.

https://unity3d.com/legal/as_terms