UnityOSC

UnityOSC v1.0 May 2011
Open Sound Control classes
and API interface for Unity 3d

Author: Jorge Garcia Martin
Email: info@jorgegarciamartin.com

1. Introduction

Open Sound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology. Bringing the benefits of modern networking technology to the world of electronic musical instruments, OSC's advantages include interoperability, accuracy, flexibility, and enhanced organization and documentation [1].

UnityOSC implements the 1.0 specification of the protocol [2] over UDP transport, except of Bundles time-stamps. This library is released under the GNU Lesser General Public License [3].

[1] http://opensoundcontrol.org/introduction-osc
[2] http://opensoundcontrol.org/spec-1_0
[3] http://www.gnu.org/licenses/

2. Architecture

The UnityOSC namespace comprises the core of the library with the following classes: OSCBundle, OSCMessage, OSCPacket, OSCClient and OSCServer. They don't include dependencies with the Unity API, so you can use them apart if you want. Take into account that they are based on the mono .NET 2.0 implementation.

OSCHandler is a proxy class that serves as tracker of the defined clients and servers. It includes a declaration of two structs, ServerLog and ClientLog, that act as containers for user-defined length logs, including date and time for each of the OSC messages. You can also use them to track incoming/outgoing packets for debugging purposes. At the moment, allowed data types that can be sent and received are: integer, long, float, double, string and byte.

Finally, OSCHelper.cs implements an User Interface for the Unity editor. It actually is a panel with buttons to monitor the logs that are generated from OSCHandler. You should include this script into the 'Editor' folder of your project. To launch the panel, simply go to Window -> OSC Helper. You have to enter the play mode in the editor to see the running servers and clients in the panel.

3. Usage

To get started, you have to define first the servers/clients that are going to be used in your project, at the OSCHandler.init() method. Any number of servers/clients can be created, but take into account that the performance would drastically decrease if many clients and servers are running at the same time, for instance, if you use multiple mobile phones or input devices. It is also possible to configure the maximum number of messages that are tracked in the logs (by default 25), by modifying the OSCHandler class attribute:

```
private const int _loglength = 25;
```

Examples of initialization usage at OSCHandler.init()

```
...
public void Init()
{
      //Initialize OSC clients (transmitters)
      CreateClient("SuperCollider", IPAddress.Parse("127.0.0.1"), 5555);

      //Initialize OSC servers (listeners)
      CreateServer("AndroidPhone", 6666);
}
...
```

Remember that the OSCHandler initialization method has to be called once at any point in your project scripts. A good place can be your Network manager or your Sound manager constructor or initialization method. Do it by just calling:

```
OSCHandler.Instance.Init();
```

You can now send messages to a defined client in two different ways. Take into a account that if the client isn't defined, OSCHandler will output a Debug error at the Unity Log console.

```
//Send a single float value to a server
float orientationAngle = 30.0f;

OSCHandler.Instance.SendMessageToClient("AServerName", "address/folder",
                                        orientationAngle);

//Send a list of object values.
//Note: you can send whatever values as a combination of floats, ints
//strings, bytes and any type allowed by UnityOSC

List<object> values = new List<object>();
values.AddRange(new object[]{1.0f, 30.0f});

OSCHandler.Instance.SendMessageToClient("AnotherServer", "address/folder",
                                        values);
```

Beware you must call first the UpdateLogs() from OSCHandler in order to read the incoming messages that are received at the servers in your application (e.g. In your Update() loop):

```
OSCHandler.Instance.UpdateLogs();
```

After the UpdateLogs() call, then you can process the received/sent messages of a defined server/client and get them by calling the properties of OSCHandler. They return a dictionary:

```
Dictionary<string, ClientLog> clients = new Dictionary<string,ClientLog>();
clients = OSCHandler.Instance.Clients;

Dictionary<string, ServerLog> servers = new Dictionary<string, ServerLog>();
servers = OSCHandler.Instance.Servers;
```

That can be useful to filter out certain messages and trigger or control events in your application, for instance, like the data output coming from the OSC mapping of a kinect device or wiimote controller.

It can be tricky to implement, but you could also dynamically create new servers or define a proxy server to allow different users connect to your application (using mobile phones, for instance).

To obtain more details about the arguments that can be passed to the methods, or how to operate with the library in other ways than the mentioned above, please head to the oxygen html documentation.

4. Support and contact

Feel free to send me an email with any issue you may have while developing with UnityOSC. If you use it in any demo or shipped product, please let me know so to include it as reference in future versions of the documentation.

Contact: info@jorgegarciamartin.com
Github repository: https://github.com/jorgegarcia/UnityOSC

Acknowledgements:
Stefan Kersten stefan.kersten@upf.edu
Paul Varcholik pvarchol@bespokesoftware.org

Donations are welcome to this PayPal account: neo_jgm@yahoo.es