SCHOOL OF MUSIC, THEATRE & DANCE
PERFORMING ARTS TECHNOLOGY
UNIVERSITY OF MICHIGAN

Creative Coding (PAT 204/504, Fall 2024)

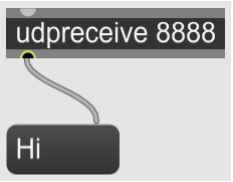
Lecture 23 – Networking & Open Sound Control

Instructor: Hao-Wen Dong

Example 1: UDP Networking ("1_udp.maxpat" & "1_udp_client.maxpat")

- Use the "udpsend" object to send data using the User Datagram Protocol (UDP). We can send different values and messages through UDP in Max.

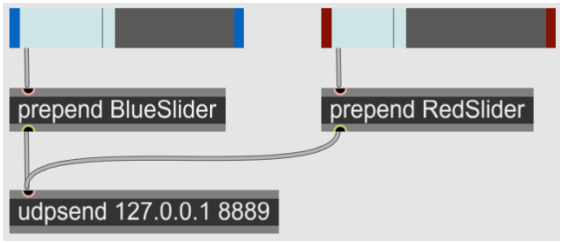
  - The second argument defines the network address to send the data to. In this example, we are using "127.0.0.1" which is a special IP address that points to the machine itself. We can also use "localhost" instead of "127.0.0.1" to send message to the machine itself.
  - The third argument defines the port number to be used to send the data to. Network ports act like PO boxes that allow routing the many messages to the correct receivers. Here, we are using 8888 as the port number. Note that some port numbers are reserved for specific important programs, so we need to avoid using those port numbers. Port numbers above 1024 are generally safe.
- Use "udpreceive" to receive data sent through UDP.



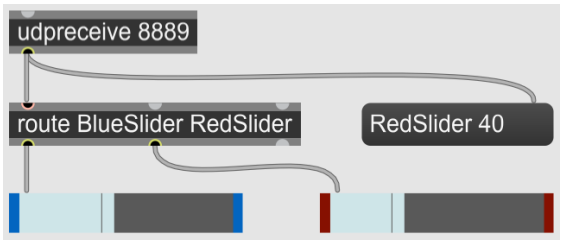
  - The second argument defines the port number to listen to. We have to use the same port number so that we can receive the data sent through the "updsend 127.0.01 8888".
  - Note that we can receive data in either the same patch or another patch (e.g., across ""1_udp.maxpat" and ""1_udp_client.maxpat").

- To better organize the data sent through UDP, we can prepend an ID to the data we are sending.

  [Sender]

  [Receiver]

  - In this example, at the sender end, we prepend “BlueSlider” to the first slider value, making it “BlueSlider 50”. Similarly, we prepend “RedSlider” to the second slider value, making it “RedSlider 50”.
  - Now, at the receiver end, we receive the data as “RedSlider 40”. We can then use the “route” object to route these messages to their corresponding sliders. Note that the “route” object will strip off the matching prefix and thus its outputs can be directly fed to the sliders here. This is different from the “select” object that won’t strip off the matching prefix.
- UDP allows us to communicate between Max to Processing (note that this is not ideal, and we will talk about a better way to communicate between Max and Processing using Open Sound Control).

```
// udp.pde

import hypermedia.net.*;

UDP udp; // An UDP object (need to install the UDP library)
String msg = "";
byte[] raw = new byte[0];

void setup() {
  size(400, 400);

  // Create an UDP object that listen to port 8888 at local host (127.0.0.1)
  udp = new UDP(this, 8888, "127.0.0.1");

  // Log the status of UDP
  udp.log(true);

  // Start listening to incoming data
  udp.listen(true);
}

void receive(byte[] data){
  // Parse the received data into a string
```

- This Processing sketch will listen to port 8888 and show the received message.

- Note the unknown characters after the message “Hi”, which are the additional bytes sent through the UDP protocol. This is undesired, and that’s why we might want to use Open Sound Control (OSC).

Example 2: OSC Max→Processing (“2_osc_max_to_processing.maxpat”)

- Open Sound Control (OSC) is an encoding protocol that defines the format of messages to be sent through other network protocols such as UDP. An OSC message consists of 1) an address pattern, 2) a type tag string, and 3) the arguments.
- In the first example, we can send an OSC-like message through UDP without much change from example 1. The only change is that the prefix starts with a slash to represent an address, e.g., “/slider/blue” and “/slider/red”.

- Even though this is not technically an OSC message, we can still treat it as if it was one in Processing, where we can use the oscP5 library to read the data:

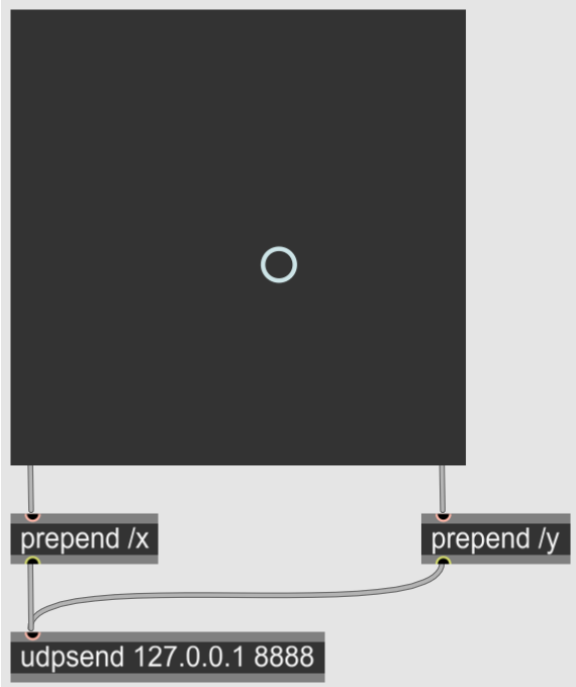
```
// osc1.pde

import oscP5.*;
import netP5.*;

OscP5 osc;
NetAddress addr;
int value = -1;

void setup() {
  size(400, 400);
  addr = new NetAddress("127.0.0.1", 8888);
  osc = new OscP5(this, 8888);
}

void oscEvent(OscMessage oscMessage) {
  // Print the address pattern and typetag of the received OSC message
  print("OSC message received");
  print(" | addrpattern: " + oscMessage.addrPattern());
  println(" | typetag: " + oscMessage.typetag());

  // Parse the integer value received
  value = oscMessage.get(0).intValue();
}

void draw() {
  background(0);
  textSize(24);
  textAlign(CENTER, CENTER);
  text(value, width / 2, height / 2);
}
```

  - The `oscEvent()` function will be triggered when there is an incoming message.

    ```
    OSC message received | addrpattern: /slider/blue | typetag: i
    ```

  - We can parse the data using `oscMessage.get(0).intValue()` since we know the data type of the main data content. (OSC fixes this issue by sending a type tag string that indicates the data type of the main data content.)
  - Note that the communication between Processing and Max is still based on the UDP protocol. However, the message sent now has follows a specific format defined by OSC.

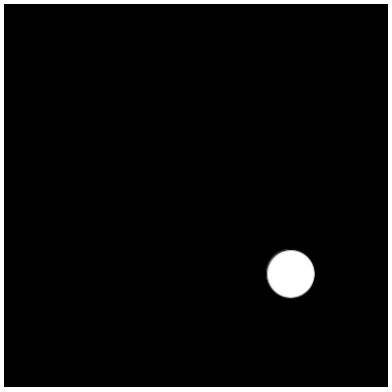
- In the next example, we want to pass a 2D slider output to Processing.




```
// osc2.pde

import oscP5.*;
import netP5.*;

OscP5 osc;
NetAddress addr;
int x = 0, y = 0;

void setup() {
  size(400, 400);
  addr = new NetAddress("127.0.0.1", 8888);
  osc = new OscP5(this, 8888);
  osc.plug(this, "setX", "/x");
  osc.plug(this, "setY", "/y");
}

void setX(int data) {
  x = data;
}

void setY(int data) {
  y = height - data;
}

void draw() {
  background(0);
  circle(x, y, 50);
}
```

    - Here, we use the `osc.plug()` function to link OSC message with a matching address pattern so that it will trigger a specific function. For example, after

running `osc.plug(this, "setX", "/x")`, whenever we receive an OSC message starting with "`/x`", the `setX()` function will be triggered.
    - This example shows how we can control Processing through Max via OSC and UDP.

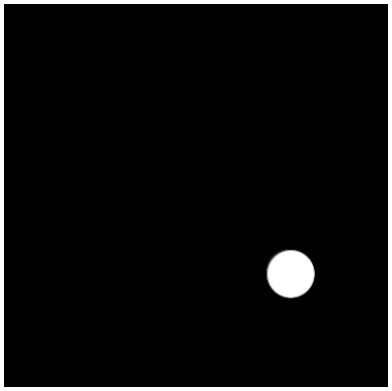


- To send a complete OSC message in Max, we will need to use the Odot package by CNMAT, which can be installed via the package manager at the left panel (icon shown below).

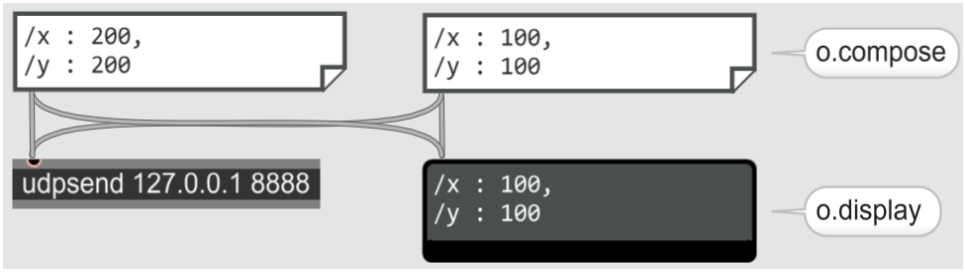
- Use "o.pack" to create an OSC bundled messages. In the following example, we create an OSC bundle with two OSC messages "/x : 103" and "/y : 287".

    - Note that the OSC message still needs to be sent through the UDP protocol.
    - The "o.display" object provides an easy way to inspect the OSC message.

- We can show the circle in Processing based on the values sent from Max.



- Use “o.compose” to manually create an OSC message

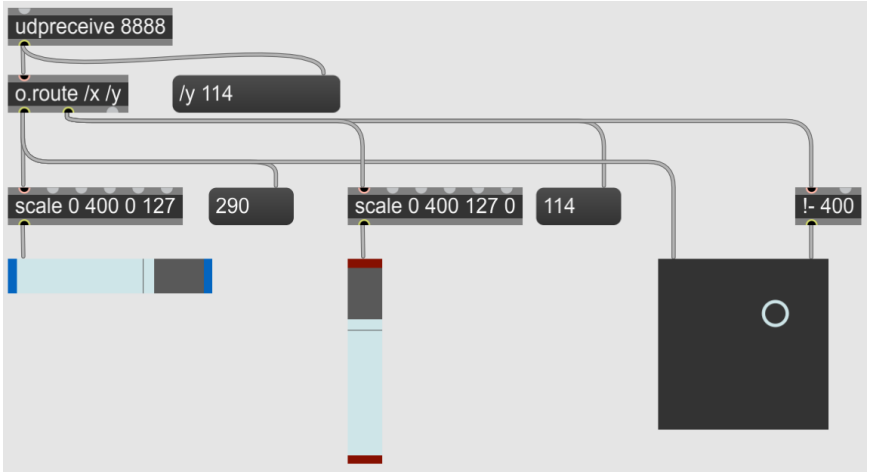


Example 3: OSC Processing→Max (“3_osc_processing_to_max.maxpat”)

- In this example, we want to send OSC messages from Processing to Max.

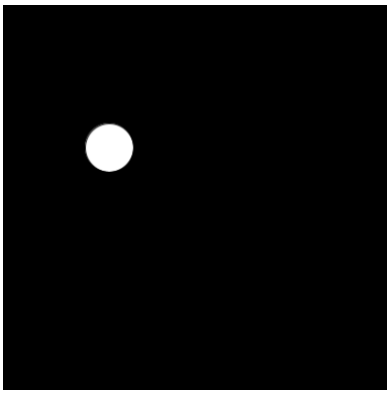
```
// osc1.pde

import oscP5.*;
import netP5.*;

OscP5 osc;
NetAddress addr;
int x = 0, y = 0;

void setup() {
  size(400, 400);
  addr = new NetAddress("127.0.0.1", 8888);
  osc = new OscP5(this, 8888);
}

void mouseMoved() {
  OscMessage mesX = new OscMessage("/x");
  OscMessage mesY = new OscMessage("/y");

  // Add the x- and y-positions to the messages
  mesX.add(mouseX);
  mesY.add(mouseY);

  // Send the messages
  osc.send(mesX, addr);
  osc.send(mesY, addr);
}

void draw() {
  background(0);
  circle(mouseX, mouseY, 50);
}
```

  - This can be done by creating `OscMessage` objects and adding values to them with `OscMessage.add()` and sending the messages through `OscP5.send()`.
  - In Max, we use an “udpreceive” object to receive the messages and an “o.route” object to route OSC messages according to their address patterns.



- In the last example, we added one more function to the Processing sketch to showcase how we can send a text string to Max through OSC:

```
// Part of osc2.pde

void mouseClicked() {
  // Create an OSC message
  OscMessage mes = new OscMessage("/clicked");

  // Add the x- and y-positions to the messages
  mes.add("clicked");

  // Send the messages
  osc.send(mes, addr);
}
```



  - When we click on the Processing window, we can see that on the Max side, we have received the "/clicked clicked" message. Also, note the difference between the "o.route" and "o.select" objects, which have similar behaviors to the "route" and "select" objects.