PPEM 2020

Comunicación

Resumen

Libraría controlP5

- Controladores
- Agrupaciones de controladores
- Visualización de datos en la GUI
- Cómo guardar valores y controlar controladores

Comunicación

- Arduino (puerto serial)
- TCP, UDP
- Requests HTTP
- Websockets
- Protocolo MIDI
- OSC
- OOCSI

Arduino

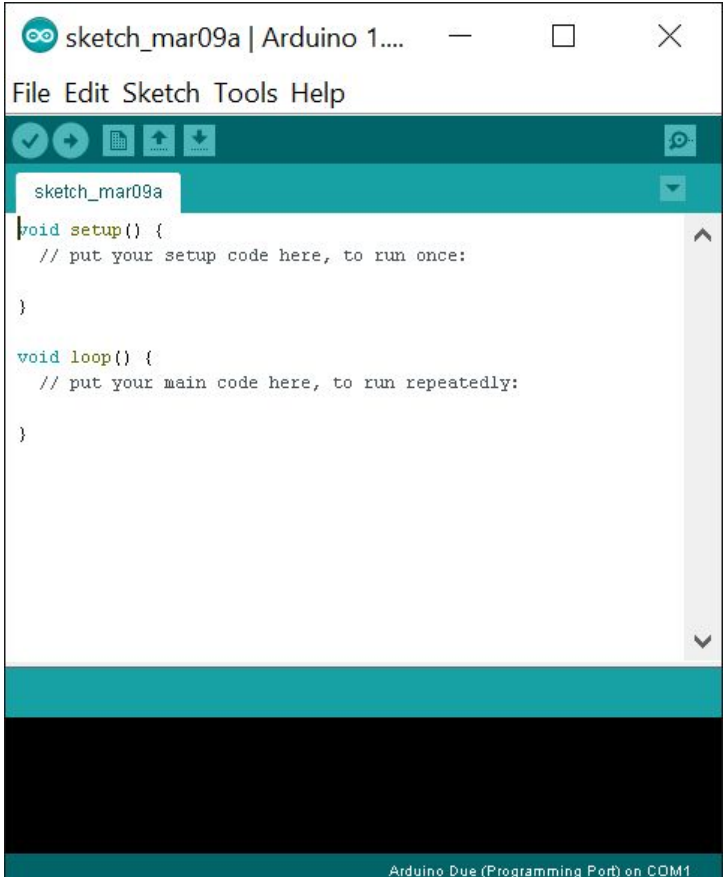
Placa computadora con su propio lenguaje de programación *Arduino Programming Language* (basado en C/C++) y su propio entorno de desarrollo *Arduino Development Environment* (basado en Processing).

- Barato
- Multiplataforma
- Entorno de programación simple y claro
- Código abierto y software extensible
- Código abierto y hardware extensible

Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Los proyectos de Arduino pueden ser autónomos o se pueden comunicar con software en ejecución en la computadora (por ejemplo con *Flash, Processing, MaxMSP*, etc.).

Ejemplo de código

```
# define LED_PIN 13
void setup () {
 // pin 13 para salida digital
 pinMode (LED_PIN, OUTPUT);
}
// Bucle infinito
void loop () {
 // Encendido del LED enviando una señal alta
 digitalWrite (LED_PIN, HIGH);
 // Tiempo de espera de 1 segundo (1000 ms)
 delay (1000);
 // Apagado del LED enviando una señal baja.
 digitalWrite (LED_PIN, LOW);
 // Tiempo de espera de 1 segundo
 delay (1000);
}
```



Processing -> Arduino

Comunicación a través del puerto serial.

Tutoriales simples de Processing y Arduino:

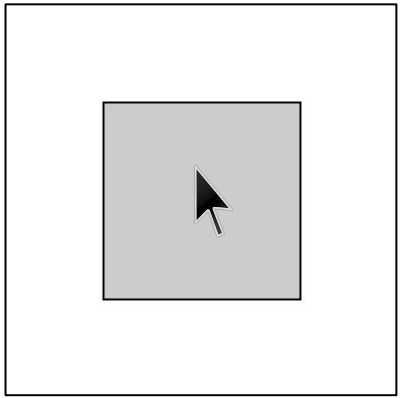
https://learn.sparkfun.com/tutorials/connecting-arduino-to-processing

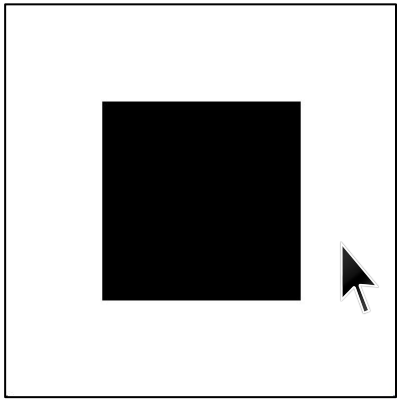
https://www.youtube.com/watch?v=7iMMUO0J3h4

Processing -> Arduino (sketch Processing)

```
import processing.serial.*;
Serial myPort;

void setup() {
  size(200, 200);
  String portName = Serial.list()[2]; // puerto usb conectado a arduino
  myPort = new Serial(this, portName, 9600);
}
void draw() {
  background(255);
  if (mouseOverRect() == true) {
    fill(204);
    myPort.write('H');
  } else {
    fill(0);
    myPort.write('L');
  }
  rect(50, 50, 100, 100);
}
boolean mouseOverRect() {
  return ((mouseX >= 50) && (mouseX <= 150) && (mouseY >= 50) && (mouseY <= 150));
}
```





Processing -> Arduino (sketch Arduino)

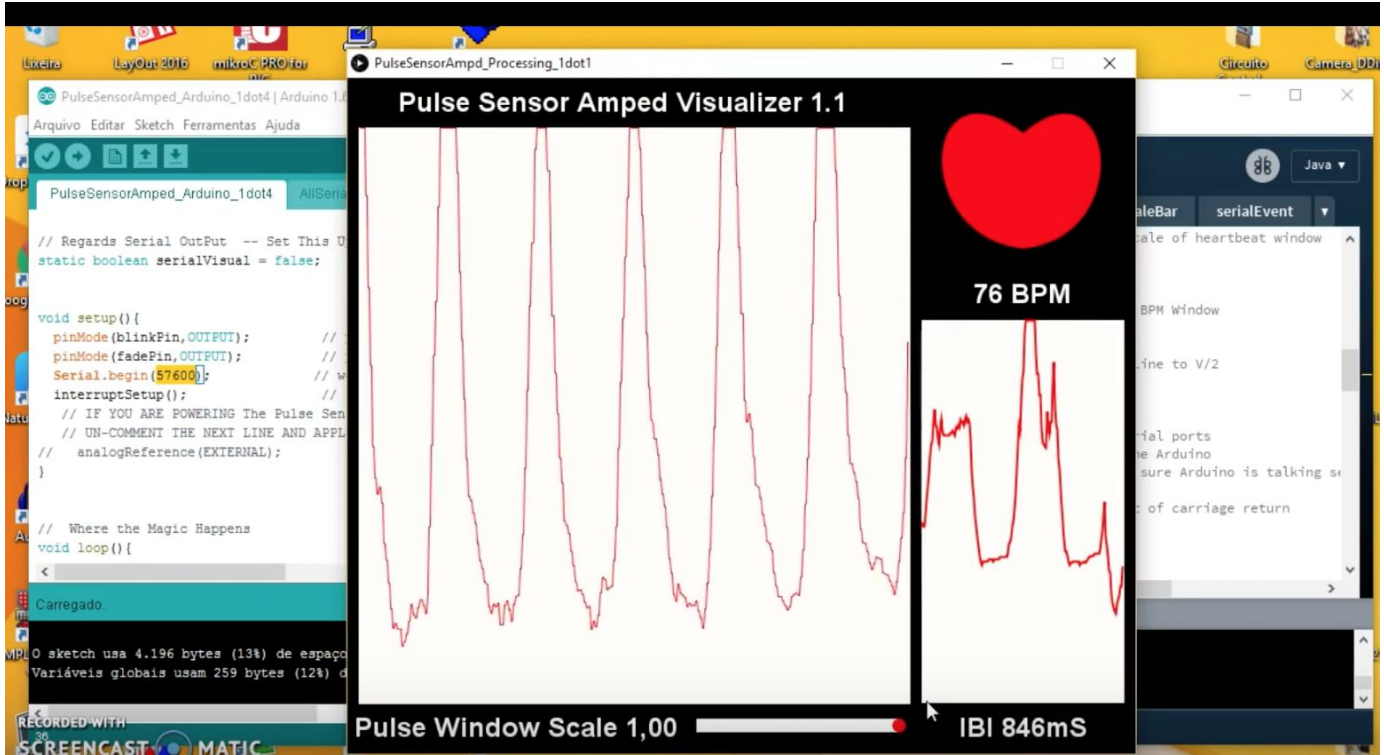
```
char val; // char que se manda desde Processing
int ledPin = 13; // el pin es el pin 13

void setup() {
  pinMode(ledPin, OUTPUT); // avisamos que va a recibir datos, no mandar
  Serial.begin(9600); // empezamos la comunicación a 9600 bits por segundo
}

void loop() {
  while (Serial.available()) { // si hay data
    val = Serial.read(); // leemos!
  }
  if (val == 'H') {
    digitalWrite(ledPin, HIGH); // prendemos el led
  } else {
    digitalWrite(ledPin, LOW); // apagamos
  }
  delay(100); // esperamos 100 milisegundos hasta próxima lectura
}
```

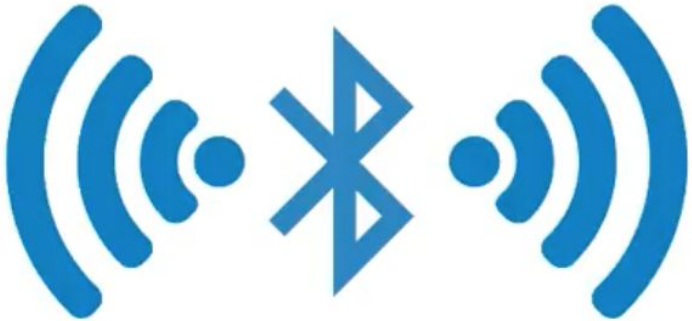
Arduino -> Processing

https://youtu.be/k_OZnCYrpoE?t=16



Arduino <-> Processing (bluetooth)

https://www.youtube.com/watch?v=VGJCj0Hr1vQ



Processing y Arduino

Juego de soplidos: https://youtu.be/H2Qs0ff6jjg?t=60

Radar: https://youtu.be/kQRYIH2HwfY?t=5

Visualización de sonido:
https://www.youtube.com/watch?v=3P7JoxfNo-0

Pintando con la luz:
https://www.youtube.com/watch?v=cYo66PIuYTE

Otras posibilidades de comunicación

Hay módulos/shields que permiten estas conexiones mediante Ethernet o WiFi y bibliotecas que resuelven el manejo de la conexión:

https://www.arduino.cc/en/Reference/WiFi

https://www.arduino.cc/en/Reference/Ethernet

Protocolo TCP y UDP

Hay dos tipos de tráfico por Protocolo de Internet (IP). Estos son **TCP** (Transmission Control Protocol) y **UDP** (User/Universal Datagram Protocol).

UDP es un protocolo **sin conexiones**. UDP **no tiene un orden** inherente y los paquetes de data son independientes uno del otro. Si requieren un orden, esto se maneja a nivel de aplicación. UDP es útil para aplicaciones que necesitan **transmisión rápida** y **efectiva**. La capacidad de transferencia sin conexiones de UDP le hace útil para servidores que reciben una gran cantidad de peticiones pequeñas de un alto número de clientes.

TCP está orientado a **conexión**. TCP es útil para aplicaciones que requieren **confiabilidad alta** y donde el tiempo de transmisión es menos crítico. TCP **reordena** paquetes de data en el orden especificado. TCP tiene **verificación de errores** y maneja **confiabilidad y control de congestión**.

TCP vs UDP

- **Slower but reliable transfers**
- **Typical applications:**
  - Email
  - Web browsing

- **Fast but non-guaranteed transfers ("best effort")**
- **Typical applications:**
  - VoIP
  - Music streaming

UDP en Processing

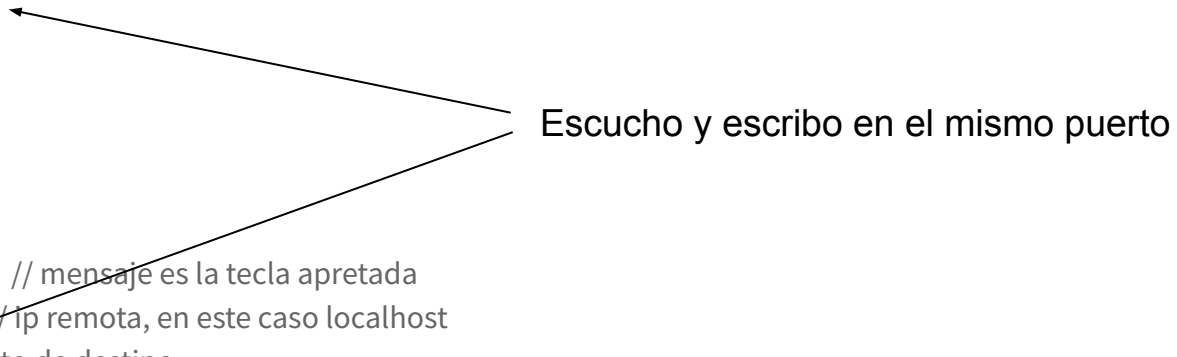
Librería UDP para Processing: http://ubaa.net/shared/processing/udp/

Streaming de imágenes por UDP sin la librería udp:

http://shiffman.net/processing.org/udp/2010/11/13/streaming-video-with-udp-in-processing/

Ejemplo UDP con la librería udp

```
import hypermedia.net.*; //udp library
UDP udp;  // define the UDP object
void setup() {
 udp = new UDP( this, 6000 );
 //udp.log( true );
 udp.listen( true );
}
void draw() {;}
void keyPressed() {
   String message  = str( key );  // mensaje es la tecla apretada
   String ip       = "127.0.0.1";  // ip remota, en este caso localhost
   int port        = 6000;   // puerto de destino
   udp.send( message, ip, port ); // se manda
}
// void receive( byte[] data ) {      // por defecto recibimos asi la data
void receive( byte[] data, String ip, int port ) {  // version extendida de la funcion de recepcion
 String message = new String( data );
 println( "Recibo: \""+message+"\" de ip "+ip+" y puerto "+port );
}
```



Ejemplo UDP sin la librería udp // envío

```
import processing.video.*;
import javax.imageio.*;
import java.awt.image.*;
import java.net.*;
import java.io.*;
int clientPort = 9100;
DatagramSocket ds;
Capture cam;
void setup() {
 size(320,240);
 try { // para DatagramSocket se precisa try/catch
  ds = new DatagramSocket();
 } catch (SocketException e) {
  e.printStackTrace();
 }
 cam = new Capture( this, width,height,30);
 cam.start();
}
void captureEvent( Capture c ) {
 c.read();
 broadcast(c); // recibimos y mandamos!
}
void draw() { image(cam,0,0);}
void broadcast(PImage img) { // imagen que vamos a mandar
 BufferedImage bimg = new BufferedImage( img.width,img.height,
BufferedImage.TYPE_INT_RGB );
 img.loadPixels();
 bimg.setRGB( 0, 0, img.width, img.height, img.pixels, 0, img.width);
 // precisamos output stream para udp
 ByteArrayOutputStream baStream      = new ByteArrayOutputStream();
 BufferedOutputStream bos            = new BufferedOutputStream(baStream);
 try {
  ImageIO.write(bimg, "jpg", bos);
 }
 catch (IOException e) {
  e.printStackTrace();
 }
 byte[] packet = baStream.toByteArray();
 try {
  ds.send(new DatagramPacket(packet,packet.length,
InetAddress.getByName("172.16.114.72"),clientPort)); // mi IP local
 }
 catch (Exception e) {
  e.printStackTrace();
 }
}
```

Ejemplo UDP sin la librería udp // recepción

```
import java.awt.image.*;
import javax.imageio.*;
import java.net.*;
import java.io.*;
int port = 9100; // puerto donde recibimos
DatagramSocket ds;
byte[] buffer = new byte[65536]; //array en cual escribimos
PImage video;

void setup() {
  size(400,300);
  try {
    ds = new DatagramSocket(port);
  } catch (SocketException e) {
    e.printStackTrace();
  }
  video = createImage(320,240,RGB);
}
 void draw() {
  checkForImage(); // ojo! bloquea, hay ejemplo con thread aparte
  background(0);
  imageMode(CENTER);
  image(video,width/2,height/2);
}

void checkForImage() {
  DatagramPacket p = new DatagramPacket(buffer, buffer.length);
  try {
    ds.receive(p);
  } catch (IOException e) {
    e.printStackTrace();
  }
  byte[] data = p.getData();
  // leemos como array de bytes
  ByteArrayInputStream bais = new ByteArrayInputStream( data );
  video.loadPixels();
  try {
    BufferedImage img = ImageIO.read(bais); // convertimos los
bytes en imagen
    img.getRGB(0, 0, video.width, video.height, video.pixels, 0,
video.width); //cargamos PImage video con pixeles recibidos
  } catch (Exception e) {
    e.printStackTrace();
  }
  video.updatePixels(); // actualizamos los pixeles en PImage video
}
```

TCP en Processing

https://processing.org/reference/libraries/net/index.html

La librería “net” permite leer y escribir datos entre computadoras a través de Internet. Permite la creación de **clientes** y **servidores**. Un servidor se conecta a una lista de clientes para leer y escribir datos. Un cliente es capaz de leer y escribir datos de/en un servidor.

La librería usa TCP para implementar las conexiones cliente - servidor.

Librería net

https://processing.org/reference/libraries/net/index.html

Client

The Client class is used to create Client objects that connect to a server to exchange data.

Client
available()
active()
read()
readChar()
readBytes()
readBytesUntil()
readString()
readStringUntil()
write()
clear()
stop()
ip()

Server

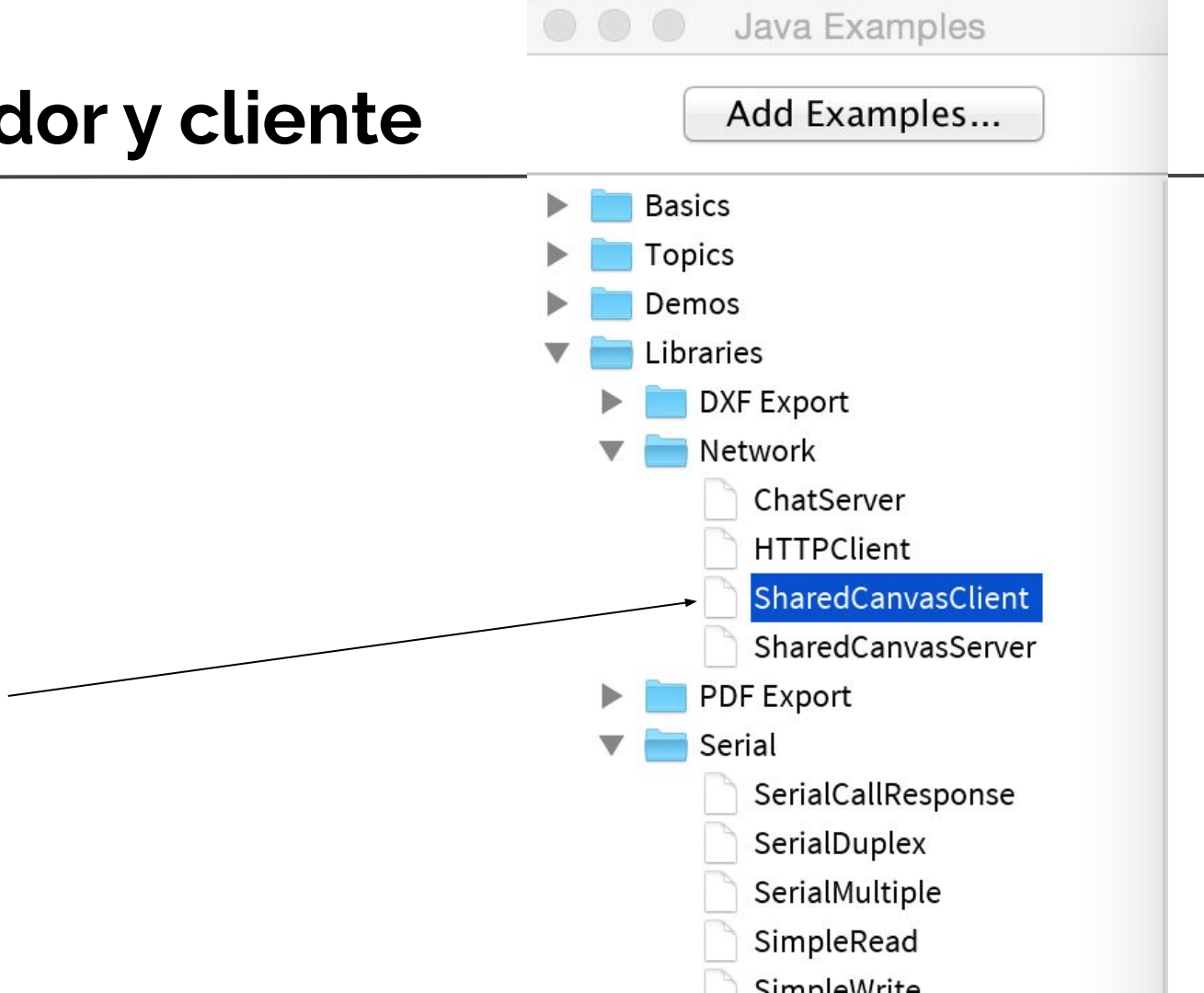
The Server class is used to create Server objects that send and receive data to and from associated clients, the other programs connected to it.

Server
write()
available()
active()
stop()
disconnect()
ip()

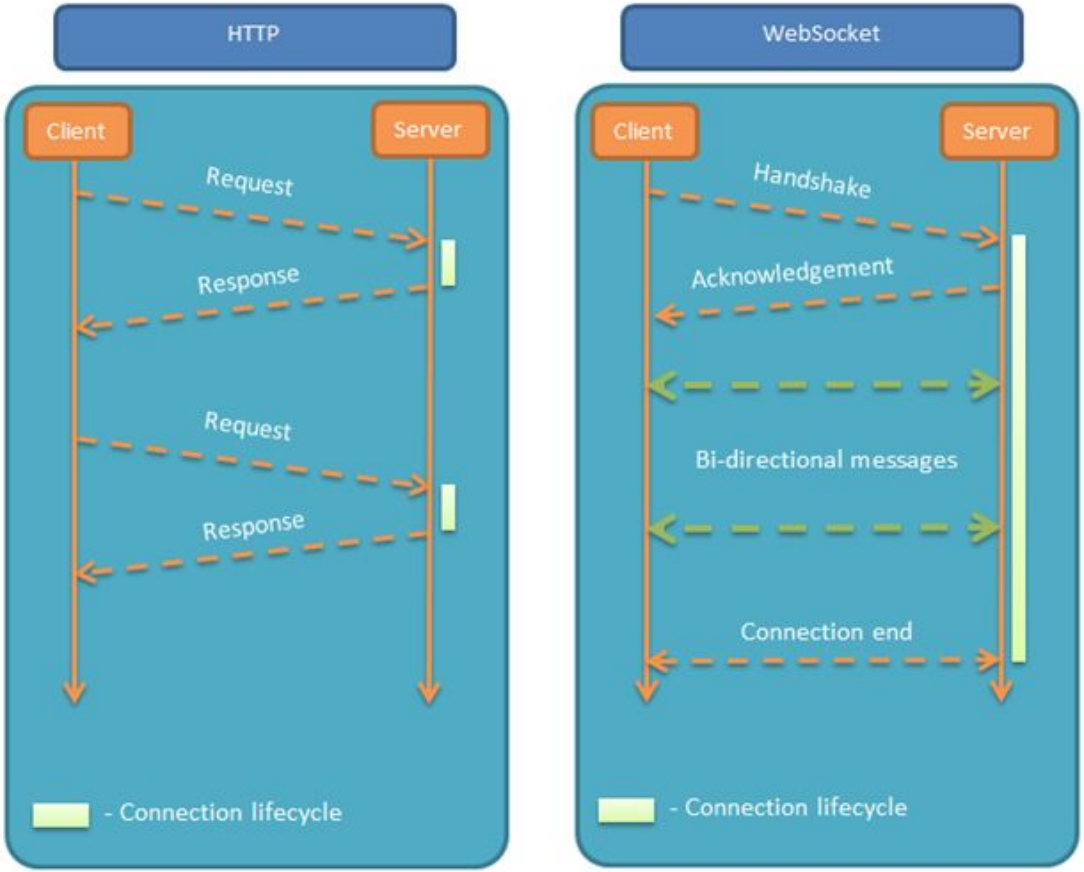
Network Events

serverEvent()
clientEvent()
disconnectEvent()

Servidor y cliente



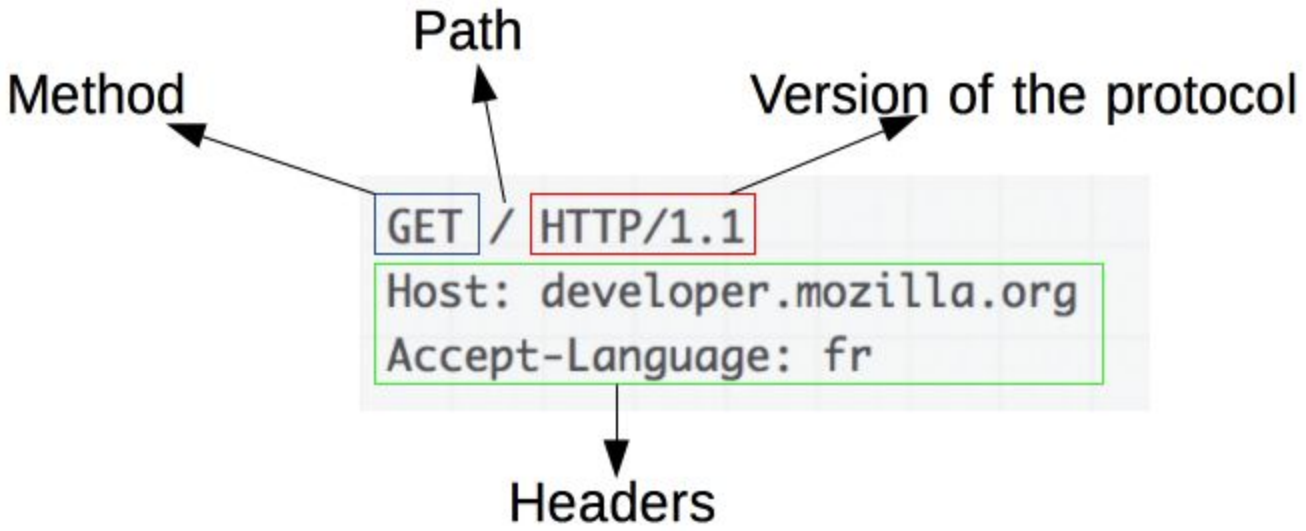
HTTP vs Websocket



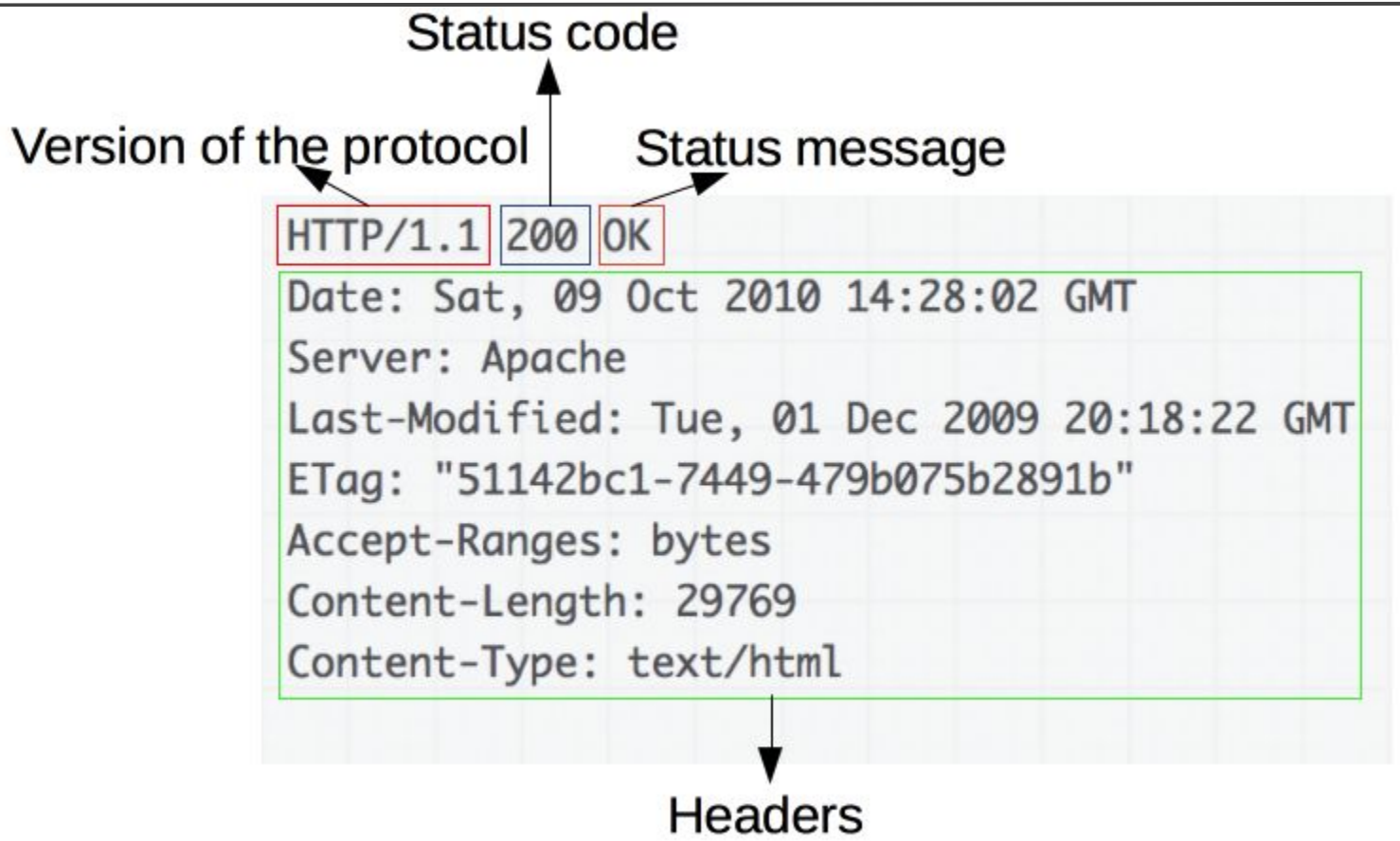
HTTP

El Hypertext Transfer Protocol (HTTP) es un protocolo de **capa de aplicación**. Funciona como un protocolo de **petición-respuesta** en el modelo de computación **cliente-servidor**. Un navegador web, por ejemplo, puede ser el cliente y una aplicación que se ejecuta en una computadora que aloja un sitio web puede ser el servidor. El cliente envía un mensaje de **solicitud HTTP** al servidor. El servidor, que **proporciona recursos** como archivos HTML y otro contenido, o realiza otras funciones en nombre del cliente, devuelve un mensaje de respuesta al cliente. La respuesta contiene información sobre el **estado de finalización** de la solicitud. Es un protocolo **sin estado** (stateless protocol) que trata cada petición como una transacción independiente que no tiene relación con cualquier solicitud anterior.

Petición


Path
Method
Version of the protocol
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
Headers

Respuesta


Status code
Version of the protocol
Status message
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
Headers

Websockets

TCP es un protocolo que especifica cómo los objetos en Internet abren, mantienen y cierran una conexión e implica múltiples intercambios de mensajes. La **conexión** que se hace entre dos objetos utilizando TCP se denomina **socket**. Es como una tubería que une los dos objetos. Permite que los datos fluyan hacia adelante y hacia atrás entre ellos mientras se mantenga la conexión. Ambos lados necesitan mantener la conexión abierta para que funcione.

Websocket es un protocolo de comunicación que proporciona un canal bidireccional de comunicaciones a través de un único socket TCP. Está diseñado para ser implementado en **navegadores web** y **servidores web,** pero puede ser utilizado por cualquier aplicación cliente o servidor.

HTTP vs Websocket

	HTTP	WebSocket
Overhead	100s of bytes	2-6 bytes (typical)
Latency	New connection each time	None: Use existing connection
Latency (polling)	Wait for next interval	No waiting
Latency (long polling)	None, if request sent earlier + time to set up next request	No waiting

https://blogs.windows.com/buildingapps/2016/03/14/when-to-use-a-http-call-instead-of-a-websocket-or-http-2-0/

TCP y ejemplo de HTTPClient

Para traer datos de la web con un get.

```
import processing.net.*;
Client c;
String data;

void setup() {
  size(200, 200);
  background(50);
  fill(200);
  c = new Client(this, "www.ucla.edu", 80); // Nos conectamos a puerto 80
  c.write("GET / HTTP/1.0\r\n"); // Usamos el comando HTTP "GET" para pedir la página web
  c.write("\r\n");
}
void draw() {
  if (c.available() > 0) { // si hay data
    data = c.readString(); // la leemos y imprimimos
    println(data);
  }
}
```

HTTP requests en Processing

https://github.com/runemadsen/HTTP-Requests-for-Processing

Obtener data del server con el GetRequest.

Mandar data a traves del PostRequest usando addData.

Autentificación usando addUser.

Agregar **headers** usando addHeaders.

Ejemplo de GetRequest

```
import http.requests.*;

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

 GetRequest get = new GetRequest("http://connect.doodle3d.com/api/list.example");
 get.send(); // el script no avanza hasta no obtener el resultado
 println("Respuesta: " + get.getContent());

 JSONObject response = parseJSONObject(get.getContent());
 println("Status: " + response.getString("status"));
 JSONArray boxes = response.getJSONArray("data");
 println("boxes: ");
 for(int i=0;i<boxes.size();i++) {
  JSONObject box = boxes.getJSONObject(i);
  println("  wifiboxid: " + box.getString("wifiboxid"));
 }
}
```

```
{
        "status":"success",
        "data":[
                {
                        "id":"62.216.8.197\/10.0.0.18",
                        "remoteip":"62.216.8.197",
                        "localip":"10.0.0.18",
                        "wifiboxid":"Albert",
                        "hidden":"0",
                        "date":"2013-10-03 17:24:33",
                },
                {
                        "id":"62.216.8.197\/10.0.0.29",
                        "remoteip":"62.216.8.197",
                        "localip":"10.0.0.29",
                        "wifiboxid":"Wilbert",
                        "hidden":"0",
                        "date":"2013-10-03 17:52:19"
                }
        ],
}
```

Websockets en Processing

https://github.com/alexandrainst/processing_websockets

La librería fue desarrollada en Mac y testeada con Processing 3.0.1

Websockets en Java en uso en Processing:

https://stackoverflow.com/questions/18900187/processing-how-to-send-data-through-websockets-to-javascript-application

WebsocketServer

```
import websockets.*;
WebsocketServer ws;
float x,y,px,py;

void setup(){
  size(200,200);
  ws= new WebsocketServer(this,8025,"/ellipse");
  x = y = px = py = 20;
}
void draw(){
  background(0);
  px = lerp(px, x, 0.05);
  py = lerp(py, y, 0.05);
  ellipse(px,py,30,30);
}
void webSocketServerEvent(String msg){
 String [] vals = msg.split(" ");
 px = x;
 py = y;
 x = int(vals[0]);
 y = int(vals[1]);
}
```

WebsocketClient

```
import websockets.*;
WebsocketClient wsc;
int x,y;

void setup(){
  size(200,200);
  x = y = 20;
  wsc= new WebsocketClient(this, "ws://127.0.0.1:8025/ellipse");
}
void draw(){
  ellipse(x,y,30,30);
}
void mousePressed(){
  x = mouseX;
  y = mouseY;
  wsc.sendMessage(x+" "+y);
}

void webSocketEvent(String msg){
 println(msg);
}
```

Protocolo Midi

Musical Instrument Digital Interface = MIDI

MIDI en sí es un estándar de comunicación serial. Se usa para comunicar datos de performance de instrumentos musicales (digitales), pero el protocolo también se ha extendido a otros dispositivos, como iluminación de escenario y equipos de estudio de grabación.

https://learn.sparkfun.com/tutorials/midi-tutorial/all

TheMidiBus

MidiBus es una biblioteca MIDI para procesamiento que proporciona una forma rápida y fácil de enviar y recibir datos MIDI. El MidiBus está diseñado principalmente para aplicaciones MIDI en **tiempo real**. Actualmente **no tiene** incorporado secuenciador, archivo de lectura/escritura, grabación MIDI/reproducción.

http://www.smallbutdigital.com/projects/themidibus/

Varios parámetros de la misma función

sendMessage(byte[] data)
Sends a MIDI message with an unspecified number of bytes.

sendMessage(int status)
Sends a MIDI message that takes no data bytes.

sendMessage(int status, int data)
Sends a MIDI message that takes only one data byte.

sendMessage(int status, int data1, int data2)
Sends a MIDI message that takes one or two data bytes.

sendMessage(int command, int channel, int data1, int data2)
Sends a channel message which takes up to two data bytes.

sendMessage(javax.sound.midi.MidiMessage message)
Sends a MidiMessage object.

Mensajes midi

Podemos **recibir**

- noteOn(int channel, int pitch, int velocity)
- noteOn(int channel, int pitch, int velocity, long timestamp, String bus_name)
- noteOff(int channel, int pitch, int velocity)
- noteOff(int channel, int pitch, int velocity, long timestamp, String bus_name)
- controllerChange(int channel, int number, int value)
- controllerChange(int channel, int number, int value, long timestamp, String bus_name)
- rawMidi(byte[] data)
- rawMidi(byte[] data, String bus_name)
- midiMessage(MidiMessage message)
- midiMessage(MidiMessage message, long timestamp, String bus_name)

Mensajes midi

Podemos **mandar**

- sendNoteOn(note)
- sendNoteOn(channel, pitch, velocity)
- sendNoteOff(note)
- sendNoteOff(channel, pitch, velocity)
- sendControllerChange(change)
- sendControllerChange(channel, number, value)
- sendMessage(message)
- sendMessage(status_byte, channel_byte, first_byte, second_byte);

Configuración del controlador midi

Midi desde celular

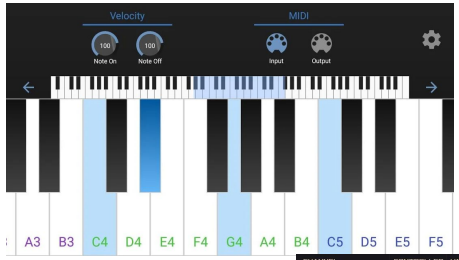
Midi Keyboard App (sólo notas) Pocket Midi (también controladores)

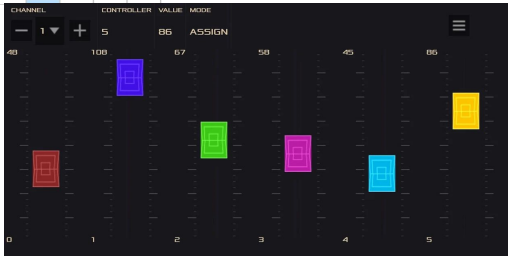
+

Cable usb entre el celular y la computadora

+

Seteo: opciones de desarrollador -> seleccionar configuración de USB -> MIDI. Gracias a estos pasos se puede setear en la aplicación la Salida en “Puerto periférico USB de Android” en Midi Keyboard App, en Pocket Midi no hay que hacer nada.





Ejemplo con controlador midi

```
import themidibus.*;
MidiBus myBus;

void setup() {
  MidiBus.list();
  //myBus = new MidiBus(this, 0, 1); // pasamos el índice para seleccionar dispositivo de entrada y salida
  myBus = new MidiBus(this,3, -1);
}

void draw() {
}

void noteOn(int channel, int pitch, int velocity) {
  println("Note On: Channel:"+channel+" Pitch:"+pitch+" Velocity:"+velocity);
}
void noteOff(int channel, int pitch, int velocity) {
  println("Note Off: Channel:"+channel+" Pitch:"+pitch+" Velocity:"+velocity);
}
void controllerChange(int channel, int number, int value) {
  println("Controller Change: Channel:"+channel+" Number:"+number+" Value:"+value);
}
```

El texto que gira con midi

```
import themidibus.*;
MidiBus myBus;
String message = "hola!mundo!con!midi!";
float rotationDegrees;
float theta;
int transX = 100;
float rotateLetter = 90;
float opacityBlack = 20;
float txtSize = 32;
void setup(){
 size(400,400);
 rotationDegrees = 360/message.length();
 theta = 0;
 background(0);
 textAlign(CENTER);
 myBus = new MidiBus(this, 3, -1);
}
void draw(){
 fill(0,opacityBlack);
 rect(0,0,width,height);
 fill(255);
 textSize(txtSize);
 translate(width/2,height/2); // nos movemos al centro
 rotate(theta); // para que haya un giro constante
 for(int i =0;i<message.length();i++){
  pushMatrix();
   rotate(radians(rotationDegrees*i)); // rotar el sistema de coordenadas
   translate(transX,0); // nos alejamos del centro
   rotate(radians(rotateLetter)); // para rotar la letra
   text(message.charAt(i),0,0); // dibujo la letra en 0,0
  popMatrix();
 }
 theta+=0.01;
}
void controllerChange(int channel, int number, int value) {
 if(number==1){
  transX = value;
 }
 else if(number==2){
  rotateLetter = map(value,0,127,0,360);
 }
 else if(number==3){
  opacityBlack = map(value,0,127,0,255);
 }
 else if(number==4){
  txtSize = map(value,0,127,32,64);
 }
}
void noteOn(int channel, int pitch, int velocity) {
 theta=0;
}
```

Mensajes midi - a quien?

Non-musical Applications

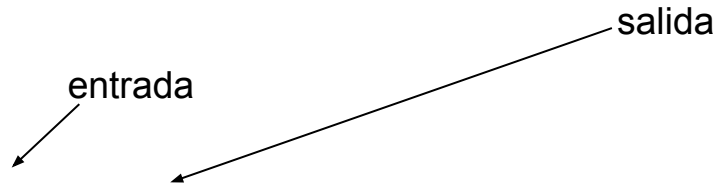
- Show control
- Machine control
- Theatre lighting
- Console automation
- Special effects
- Sound design
- Recording system synchronization
- Audio processor control
- Computer animation
- Computer networking
- Video Jockeys

Midi de Processing a Resolume

```
import themidibus.*;

MidiBus myBus;
int value = 0;
void setup() {
  myBus = new MidiBus(this, -1, "virtualPort");
}

void draw() {
  if(frameCount%5==0){
    value +=1;
    if(value > 127){
      value = 0;
    }
    // sendControllerChange(channel, number, value);
    myBus.sendControllerChange(0, 0, value);
  }
}
```



Midi por WIFI o Ethernet

WIFI: https://vimeo.com/119870372

Ethernet: http://yaskebasi.it/blog/2015/03/02/how-to-send-midi-over-ethernet-tutorial/

OSC

Open Sound Control

Es un protocolo de comunicaciones que permite comunicar instrumentos de música, computadoras y otros dispositivos multimedia pensado para compartir información musical en tiempo real sobre una red. Aparece como reemplazo del MIDI.

Características principales del protocolo:

- Ampliable, dinámico. Esquema de nombres simbólicos tipo URL
- Datos como integers, doubles, booleanos y strings
- Lenguaje de coincidencia de patrones para especificar múltiples receptores de un único mensaje
- Marcas de tiempo de alta resolución.
- Mensajes “empaquetados” para aquellos eventos que deben ocurrir simultáneamente

Puede ser transportado por varios protocolos, pero comúnmente se usa UDP.

Comparación con midi

Fuente de la gráfica:

http://opensoundcontrol.org/files/OSC-Demo.pdf

Contra de midi:
https://www.midi.org/articles/white-paper-comparison-of-midi-and-osc

	OSC	MIDI

Examples of Messages	`/wii/ir/x 0.1503`
	`/stylus/pressure 0.014`
	`/camera/look-at 5. 12. 17.`
	`/play-note 15 0.9`
		`144 60 64 (MIDI Note-on)`
		`128 60 64 (MIDI Note-off)`

Message types	User-defined Human-readable	Pre-determined 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

Áreas de aplicación

Sensor/Gesture-Based Electronic Musical Instruments

Mapping nonmusical data to sound

Multiple-User Shared Musical Control

Web interfaces

Networked LAN Musical Performance

WAN performance and Telepresence

Virtual Reality

Wrapping Other Protocols Inside OSC

Software que soporta OSC

Examples of software with OSC implementations:

- Ardour
- Bidule
- CasparCG
- ChucK
- Crystal Space
- CSound
- Cubase
- Digital Performer
- Dorico
- ENTTEC LED Mapper (E.L.M)
- Fluxus
- FreeJ
- Gesture Recognition Toolkit
- IanniX
- Impromptu
- Isadora (v.1.1)
- JUCE (Framework)
- Kyma
- LightFactory
- Lily
- LiVES
- Logelloop
- Logic Pro
- Max/MSP
- Mocolo
- Modul8
- MuseScore
- Mxwendler
- Nuendo
- OpenFrameworks
- Overtone (Clojure)
- Processing
- Pure
- Pure Data
- QLab
- Quartz Composer (as of v3.0 / Mac OS X v10.5)
- Reaktor
- REAPER
- Renoise
- Sonic Pi
- SuperCollider
- Squeak
- Stamp
- Strand NEO
- TouchDesigner
- Traktor DJ Studio
- Unreal Engine
- Veejay
- VirtualDJ
- vvvv

oscP5

Implementación de OSC para Processing.

oscP5 soporta TCP, UDP y Multicast (usando UDP).
Las operaciones de red son manejadas por la libreria netP5.
http://www.sojamo.de/libraries/oscP5/reference/netP5/package-summary.html

OSC de Processing a Processing

```
import oscP5.*;
import netP5.*;
OscP5 oscP5;
NetAddress myRemoteLocation;
void setup() {
  frameRate(25);
  oscP5 = new OscP5(this,12000);
  myRemoteLocation = new NetAddress("127.0.0.1",12000);
}
void draw() {
}
void mousePressed() {
  OscMessage myMessage = new OscMessage("/test-ppem-2020");
  myMessage.add(123); // prueben myMessage.add(123.0) o myMessage.add(“123”);
  oscP5.send(myMessage, myRemoteLocation);
}
void oscEvent(OscMessage theOscMessage) {
  println("Mensaje OSC: addrpattern: "+theOscMessage.addrPattern()+" typetag: "+theOscMessage.typetag());
}
```

OSC bundle

```
import oscP5.*;
import netP5.*;
OscP5 oscP5;
NetAddress myRemoteLocation;
void setup() {
  oscP5 = new OscP5(this,12000);
  myRemoteLocation = new NetAddress("127.0.0.1",12000);
}
void draw() {
}
void mousePressed() {
  OscBundle myBundle = new OscBundle();
  OscMessage myMessage = new OscMessage("/test");
  myMessage.add("abc");
  myBundle.add(myMessage);
  myMessage.clear();
  myMessage.setAddrPattern("/test2"); // un mensaje más
  myMessage.add("defg");
  myMessage.add(1234);
  myBundle.add(myMessage);
  myBundle.setTimetag(myBundle.now() + 10000);
  oscP5.send(myBundle, myRemoteLocation);
}
```

```
void oscEvent(OscMessage theOscMessage) {
  println("== Recibi: addrpattern: "+theOscMessage.addrPattern()+"
typetag: "+theOscMessage.typetag()+" timetag:
"+theOscMessage.timetag());
}
```

OSC de Processing a Resolume

```
import oscP5.*;
import netP5.*;
OscP5 oscP5;
NetAddress myRemoteLocation;

void setup() {
 size(400,400);
 frameRate(30);
 oscP5 = new OscP5(this,12000);
 myRemoteLocation = new NetAddress("127.0.0.1",7000);
}

void draw() {
 if(frameCount%2==0){
  OscMessage myMessage = new OscMessage("/activeclip/video/effect1/param2/values");
  myMessage.add(map(mouseY,0,height,0,1)); // agrego float al mensaje
  oscP5.send(myMessage, myRemoteLocation);
 }
 else{
  OscMessage myMessage = new OscMessage("/activeclip/video/effect2/param6/values");
  myMessage.add(map(mouseX,0,width,0,1)); // agrego float al mensaje
  oscP5.send(myMessage, myRemoteLocation);
 }
}
```

OSC por wifi de mi casa

```
import oscP5.*;
OscP5 oscP5;
float x = 0;
float y = 0;

float backColor = 0;

void setup() {
  size(400, 400);
  oscP5 = new OscP5(this,7000); //escucho en 7000
  colorMode(HSB);
}

void draw() {
  background(backColor,255,255);
  fill(255,0,255);
  rect(x,y,100,100);
}
```

Controlado por OSC

Controlado por OSC

```
void oscEvent(OscMessage theOscMessage) {
  print(" addrpattern: "+theOscMessage.addrPattern());
  println(" typetag: "+theOscMessage.typetag());
  if(theOscMessage.addrPattern().equals("/oscControl/slider2Dy") )
    y = theOscMessage.get(0).floatValue();
  else if (theOscMessage.addrPattern().equals("/oscControl/slider2Dx"))
    x = theOscMessage.get(0).floatValue();
  else if (theOscMessage.addrPattern().equals("/oscControl/color"))
    backColor = theOscMessage.get(0).floatValue();
}
```

OSC Controller es la app que usé

Processing, OSC y...

OSC desde celular: https://youtu.be/NBP5kdbQGdw?t=41

Processing y PureData: https://youtu.be/xQObbtbZZw0?t=58

Processing y Animata: https://vimeo.com/51966184

Processing y Resolume: https://andreasrefsgaard.dk/project/processing_resolume_fft/

OOCSI

Out of Control for Semantic Interactivity (OOCSI) es un **framework** de comunicación **multiplataforma** que permite una fácil mensajería entre los clientes. OOCSI es ligero, orientado a mensajes, permite la mensajería punto a punto y la difusión mediante los **canales** (o temas).

Hay dos componentes básicos que forman una red OOCSI: el cliente y el servidor. Esta plataforma puede conectar ordenadores Windows, Mac y Linux (corriendo Java y Processing), dispositivos (Arduino, Raspberry Pi y Gadgeteer), navegadores web (vía websockets) y dispositivos móviles (iOS y Android).

http://iddi.github.io/oocsi-processing/

https://github.com/iddi/oocsi

https://github.com/iddi/oocsi/wiki/OOCSI-Protocol

Que permite

Mensajes entre servidor y clientes

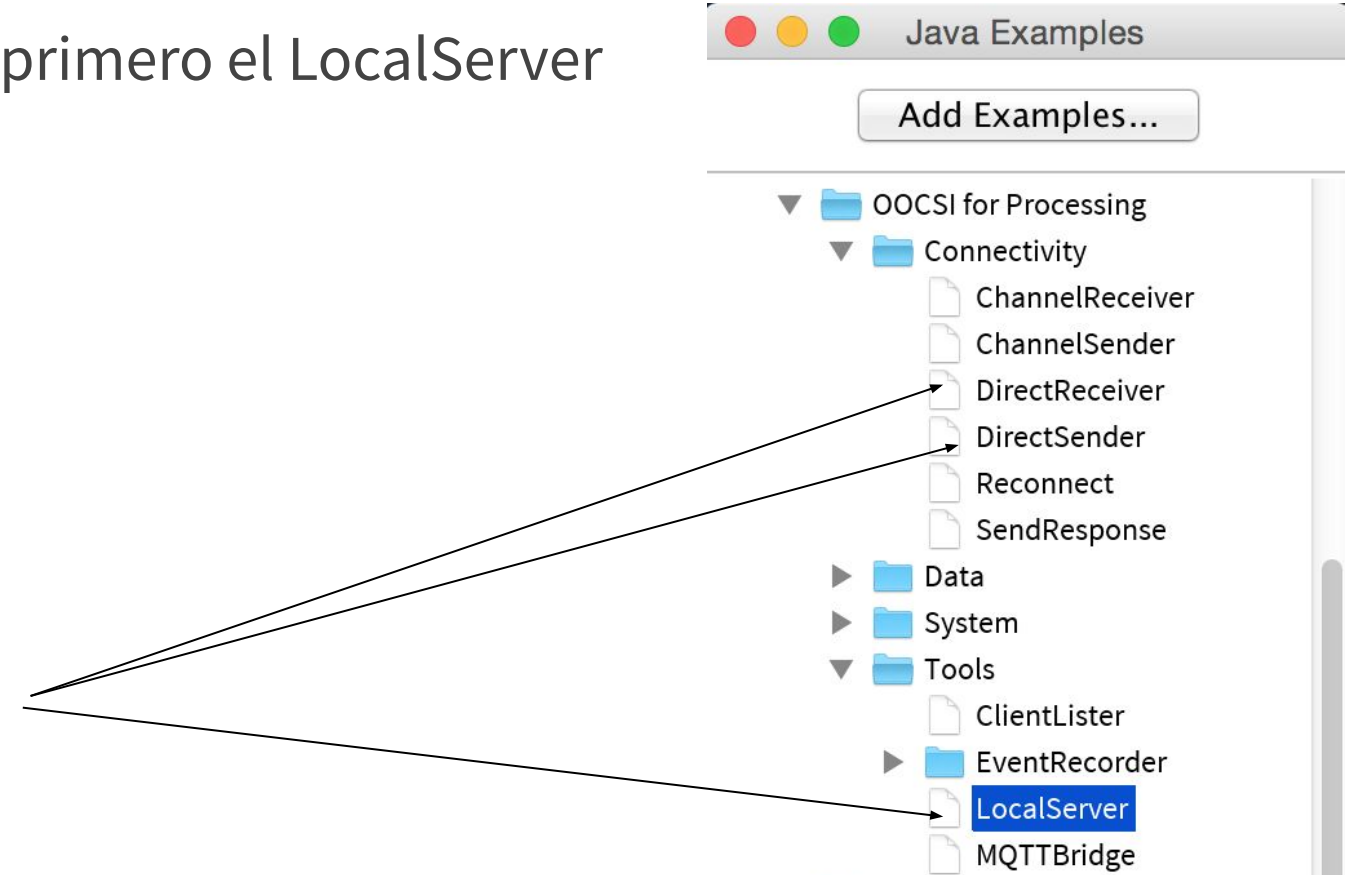
Mensajes key-value (int, float, double, long, string, objects, arrays)

Canales (suscribirse, mandar mensaje)

Reconexión

DirectSender y DirectReceiver

Hay que correr primero el LocalServer



ChannelSender y ChannelReciver

Hay que correr primero el LocalServer