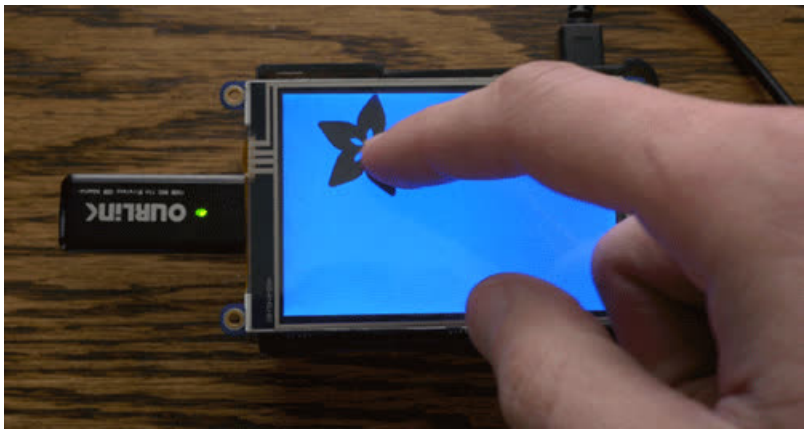
adafruit learning system

# Using OSC to Communicate with a Raspberry Pi

Created by Todd Treece




Last updated on 2018-10-09 09:39:58 PM UTC

## Guide Contents

## Overview

If you are a regular user of audio/visual programming environments like Cycling '74's Max, Pure Data, or ChucK, you may have encountered instances where you would like to communicate with remote devices and hardware. There are plenty of ways you could approach this, but a protocol known as Open Sound Control (https://adafru.it/eUg) (OSC) is probably the best supported solution.

Here's a short description of OSC from opensoundcontrol.org:

> 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.

In this tutorial, we will be exploring using OSC to communicate with a Raspberry Pi connected to your local network using Max, Pure Data, and ChucK. We will be using Node.js (https://adafru.it/eUh)on the Pi because the event driven nature of Node.js makes it easy to do things like listen to incoming OSC messages or button presses while performing other tasks. If you would like to learn more about using Node.js with your Raspberry Pi, check out our guide (https://adafru.it/iDR) for more info.

This tutorial assumes you have some basic knowledge of using the command line. If you need help to get up to speed, check out this great guide (https://adafru.it/slE) by Brennen. If you feel comfortable using the command line, then you are ready to proceed with setting up your Raspberry Pi.

## Setting Up Your Raspberry Pi

The first step in setting up your Raspbery Pi is to install the latest version of Raspbian on your SD card using NOOBS or by copying the Raspbian image directly to your card.  If you need help installing Raspbian, check out our SD card preparation guide  (https://adafru.it/Cfo)for help.

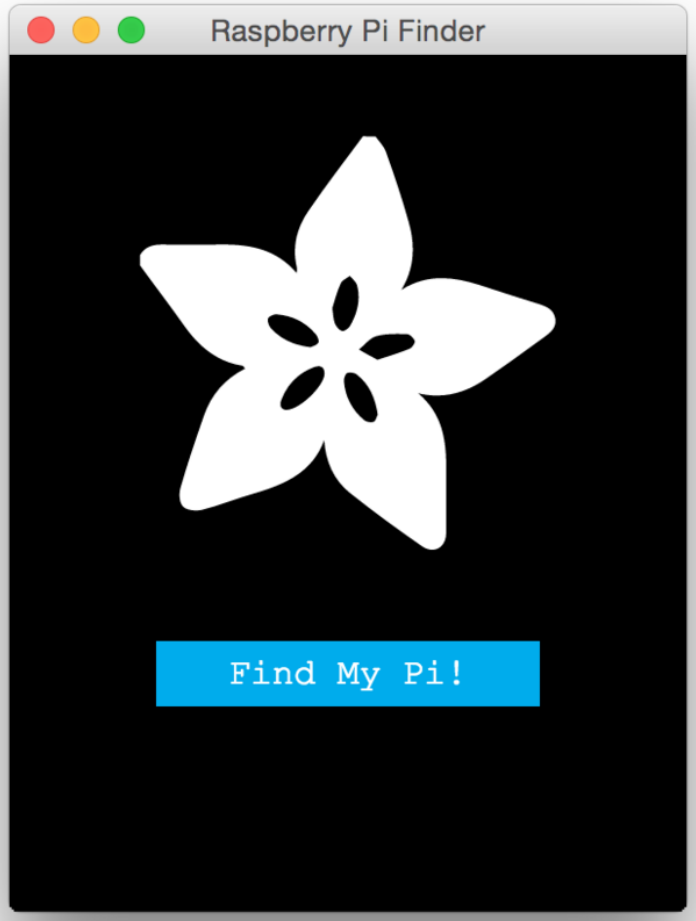
### Connecting to Your Raspberry Pi via SSH

The next step will be to connect your Raspberry Pi to your local network using either an ethernet cable or WiFi. The easiest way to get started is by connecting your Pi via the ethernet port, and using the Adafruit Pi Finder (https://adafru.it/enj) to connect to it.

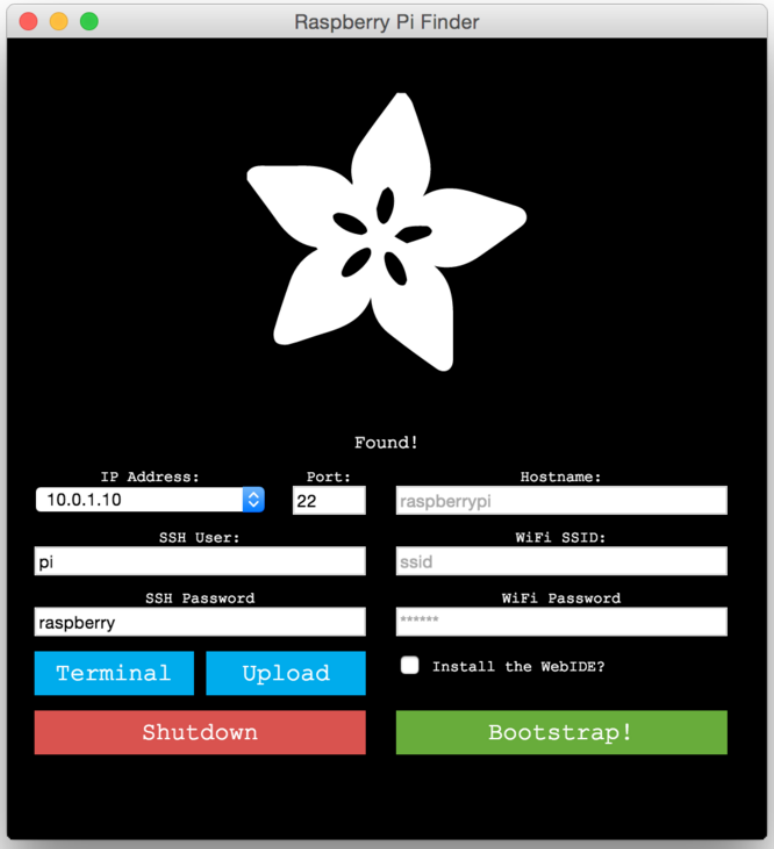
https://adafru.it/euj

https://adafru.it/euj

Once your Pi is plugged in to the local network, you can use the Pi Finder to find the IP address of your Pi by clicking **Find My Pi!**



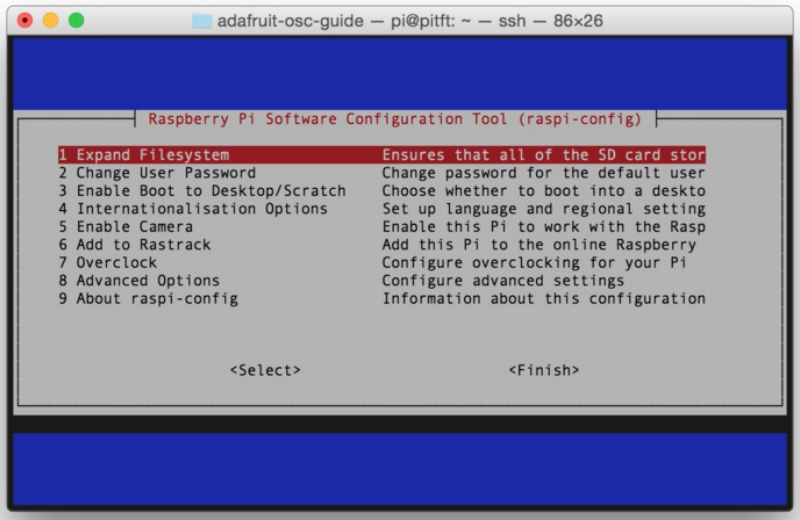

Once the Pi Finder finds the IP Address of your Pi, you can connect to it via SSH using the **Terminal** button, or by using your favorite SSH client.




## Expanding the Filesystem

Once you have connected to your Pi, you will need to expand the filesystem using **sudo raspi-config** if you haven't done so already.

```
$ sudo raspi-config
```




Once you have expanded the filesystem, click *Finish* and *Yes* to restart the Pi.

## Installing Node.js

Now we are ready to install the latest version of Node.js. See the instructions on our basic Node.js embedded development guide (https://adafru.it/iDR).

Next, you can install the **libcairo2-dev** package:

```
$ sudo apt-get install libcairo2-dev
```

## Downloading the OSC Examples Repository

Next, we will use **git** to download the latest version of the OSC example repository we will be using for this tutorial.

```
$ git clone https://github.com/toddtreece/osc-examples.git
```

Then **cd** into the *osc-examples* directory.

```
$ cd osc-examples
```

Finally, we will need to install the dependencies required to run the example Node.js scripts. You can do this by running **npm install**.

```
$ npm install
```

Next, we will be looking at how to set up your computer with Max, Pure Data, or ChucK.

## Setting Up Your Workstation

To run the examples on your computer, you will need a copy of the latest version of Cycling '74's Max, PureData (Pd-extended), or ChucK. If you are already familiar with one of the three, then the choice will be obvious, but please make sure you are using a recent version.

### Cycling '74 Max

Max is a visual programming environment that enables musicians, visual artists, and researchers to create software without writing lines of code. My favorite of the three choices is Max because it was my first introduction to any type of programming, but Max is commercial software. You can download and use Max free for 30 days if you would like to try it. View the video below for more info about getting started with Max 7.

https://adafru.it/eUj

https://adafru.it/eUj

### Pure Data

Pure Data (Pd) is an open source visual programming language that is very similar in scope and design to the original version of Max. It runs on Linux, Mac OS X, iOS, Android and Windows, and is available for free download.

https://adafru.it/eUk

https://adafru.it/eUk

### ChucK

ChucK is a real-time programming language that is very different from Max and Pure Data, but has a lot of the same capabilities. The biggest difference is that ChucK is not a visual programming language like Max or PureData. ChucK is available for free, and runs on Max OS X, Linux and Windows. The easiest way to run ChucK is by using the miniAudicle IDE available on ChucK's website.

> ChucK presents a unique time-based, concurrent programming model that's precise and expressive (we call this strongly-timed), dynamic control rates, and the ability to add and modify code on-the-fly. In addition, ChucK supports MIDI, OpenSoundControl, HID device, and multi-channel audio. It's fun and easy to learn, and offers composers, researchers, and performers a powerful programming tool for building and experimenting with complex audio synthesis/analysis programs, and real-time interactive music.

https://adafru.it/eUl

https://adafru.it/eUl

### Examples

There are versions of the examples available for all three environments, and you can download the latest version of the examples by clicking the link below.

https://adafru.it/eUm

https://adafru.it/eUm

## Testing Communication

Now that you have both environments ready, you are ready to test out basic communication. Connect to your Pi via SSH again, and navigate to the **osc-examples** folder.

```
$ cd osc-examples
```

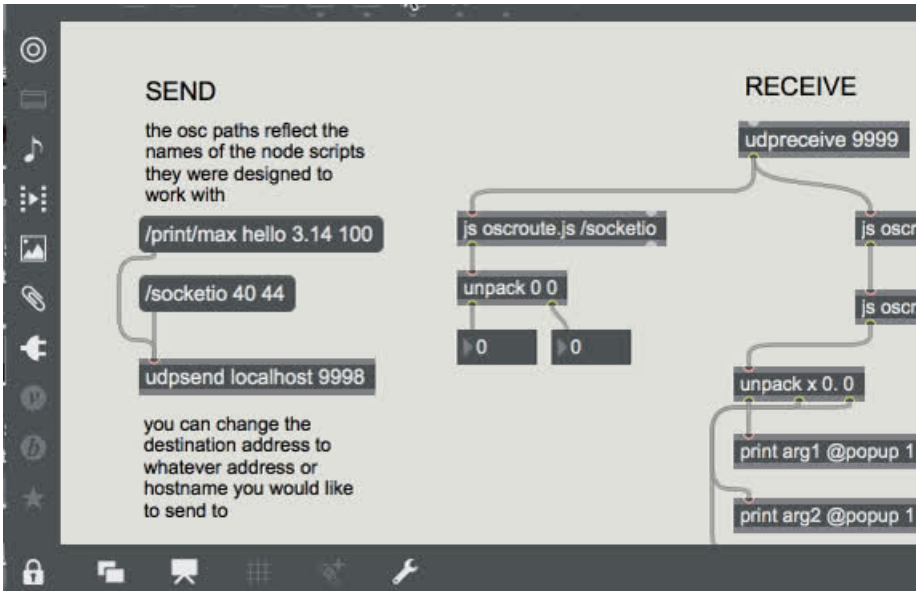
Now you can start the print example by running **node print.js**.

```
$ node print.js
```

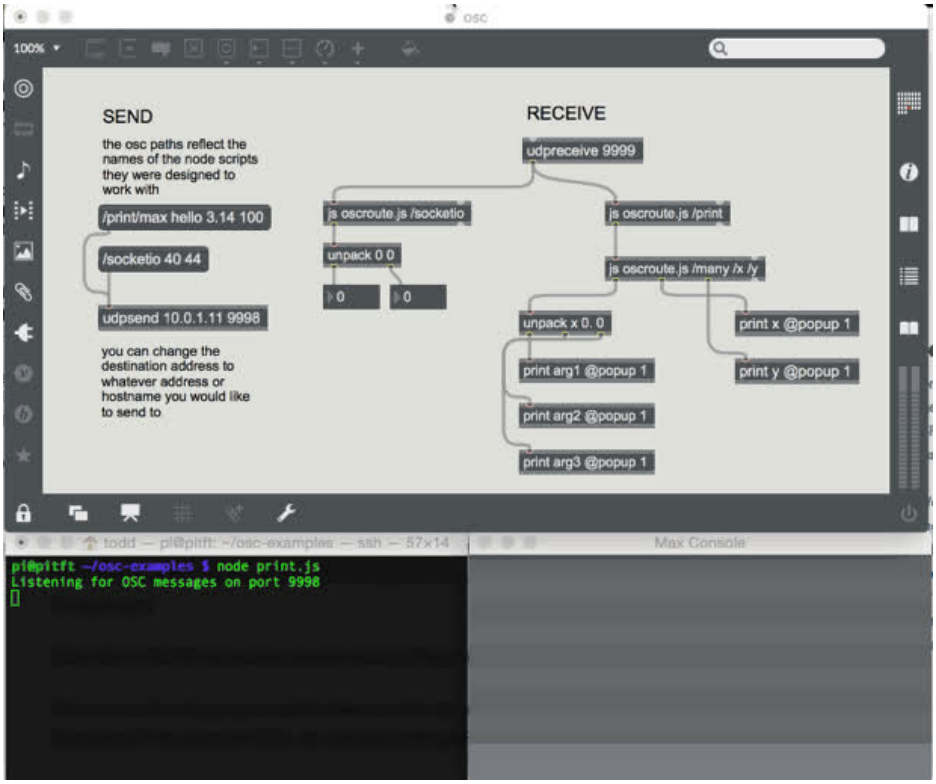
You should see output that looks like this in the terminal window. Leave this window open while testing with your choice of Max, Pure Data, or ChucK.


### Max 7

To test with Max, you will need to open the *osc* patch inside the *max* examples folder on your computer. Once opened, click the lock button to enter edit mode, and edit the **udpsend** box to replace *localhost* with the IP Address of your Pi.



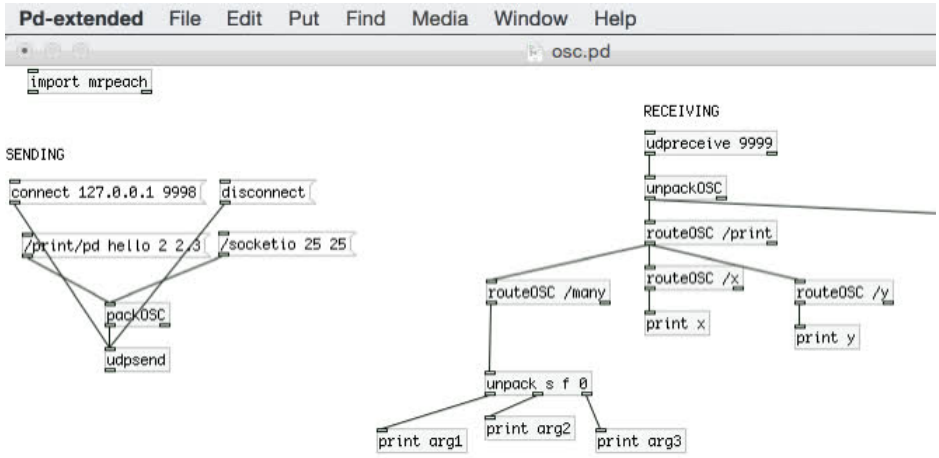

Once you have that completed, click the lock button to exit edit mode and click on the **/print/max** message box. You should see a message appear in your Raspberry Pi SSH window.



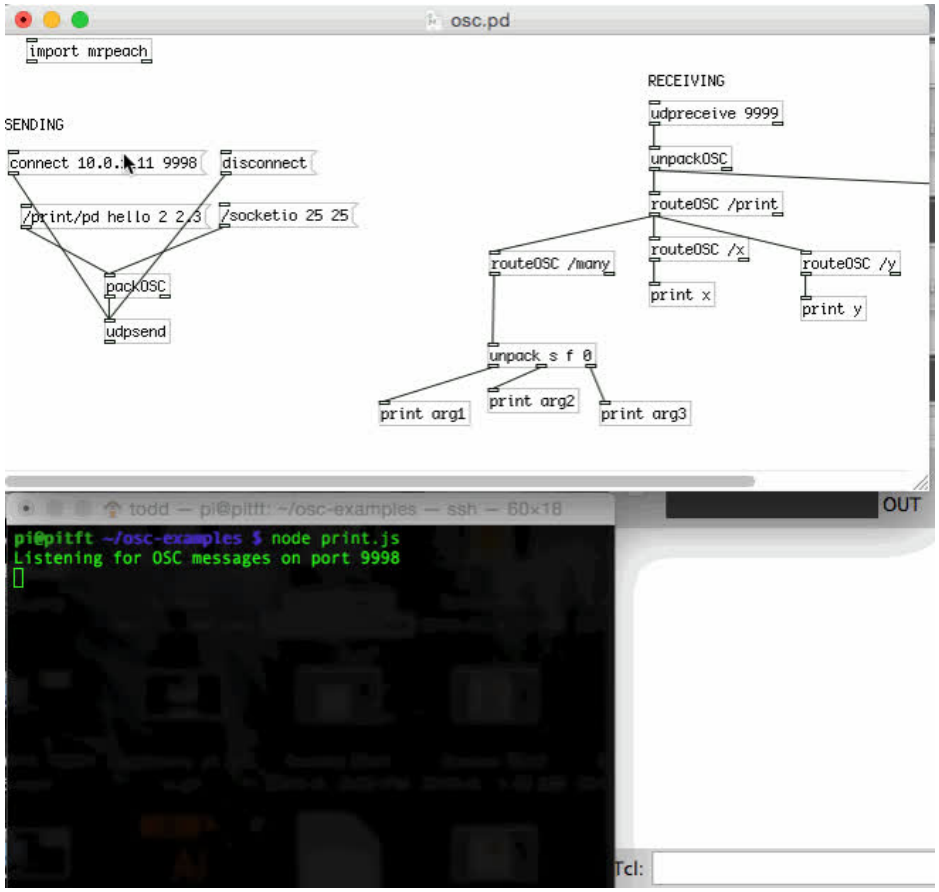

If everything is working as expected, you will see output in both the Max window and the Raspberry Pi terminal. The Node.js script will send test messages once every second to Max using OSC. You can see that the test messages include strings, floats, and integers. The **oscroute.js** helper object is used to route OSC message paths, and the **unpack** object is used to route the OSC arguments sent. You can use the arguments just like any other data source in in Max.

## Pure Data

To test with Pure Data, you will need to open the *osc* patch inside the *pd* examples folder on your computer. Once opened, enter edit mode by selecting it from the Edit menu. Change the IP Address in the *connect* object box to match the IP address of your Pi.




Once you have changed the IP, exit edit mode and click on the **/print/pd** message box to test sending and receiving data.

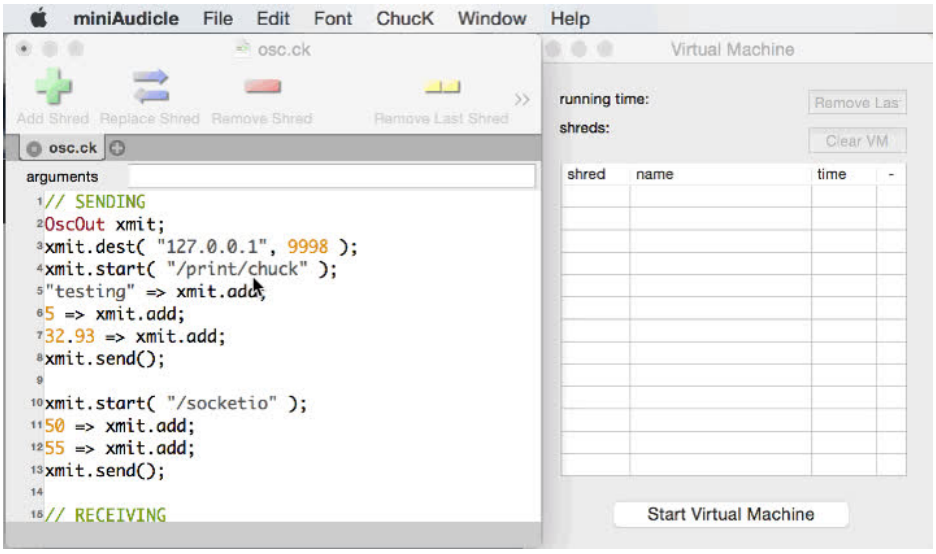



If everything is working as expected, you will see output in both the Pd window and the Raspberry Pi terminal. The

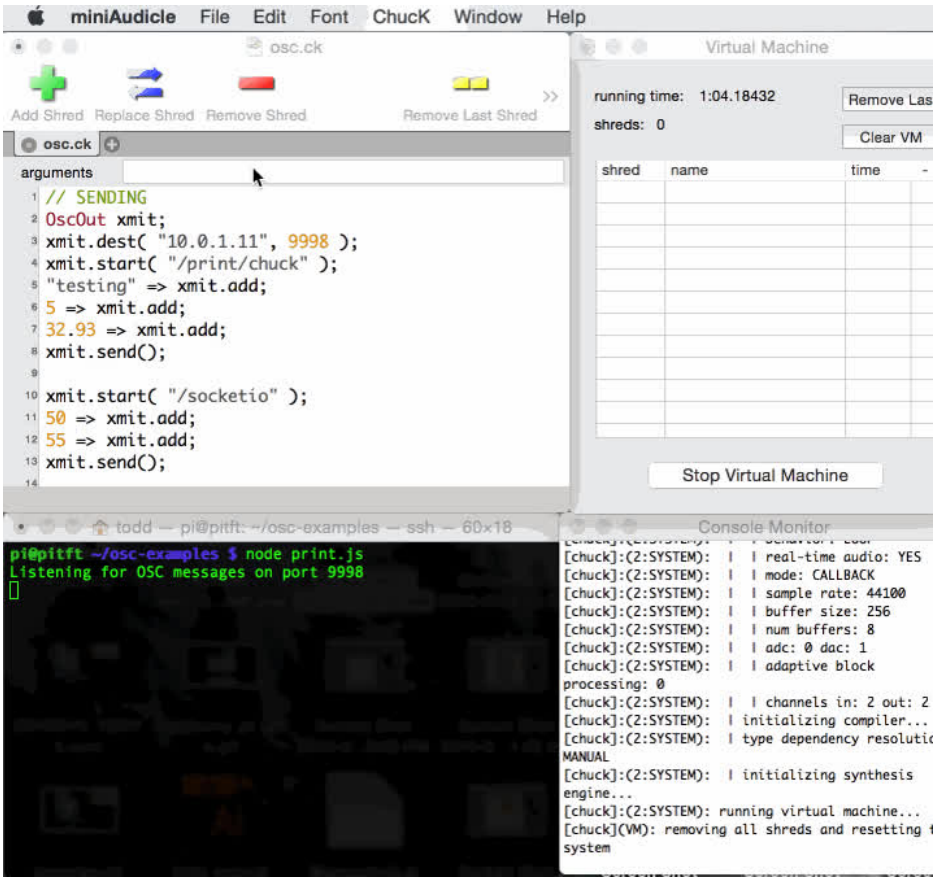
Node.js script will send test messages once every second to Pure Data using OSC. You can see that the test messages include strings, floats, and integers. The **routeOSC** object is used to route OSC message paths, and the **unpack** object is used to route the OSC arguments sent. You can use the arguments just like any other data source in in Pure Data.

## ChucK

To test with ChucK, open up the *osc.ck* script in the miniAudicle IDE. Edit the IP address to match the IP address of your Pi, and click *Start Virtual Machine*.




Click *Add Shred* to send a test message to your Pi, and to start listening for new OSC messages from the Pi.




If everything is working as expected, you will see output in both the miniAudicle console monitor and the Raspberry Pi terminal. The Node.js script will send test messages once every second to ChucK using OSC. The test messages include strings, floats, and integers. You can use the sent data just like any other data source in in ChucK.

## Real-Time Interaction with Web Browsers

In this example, we will be using Node.js and Socket.io (https://adafru.it/egq) on the Raspberry Pi to communicate in real-time with web browsers. How could this be used? Let's say you are an artist who uses Max, Pure Data, or ChucK for live performances, and want the audience to be able to interact with your performance. You could use this method to allow audiences to change parameters from the web browsers on their smart phones.

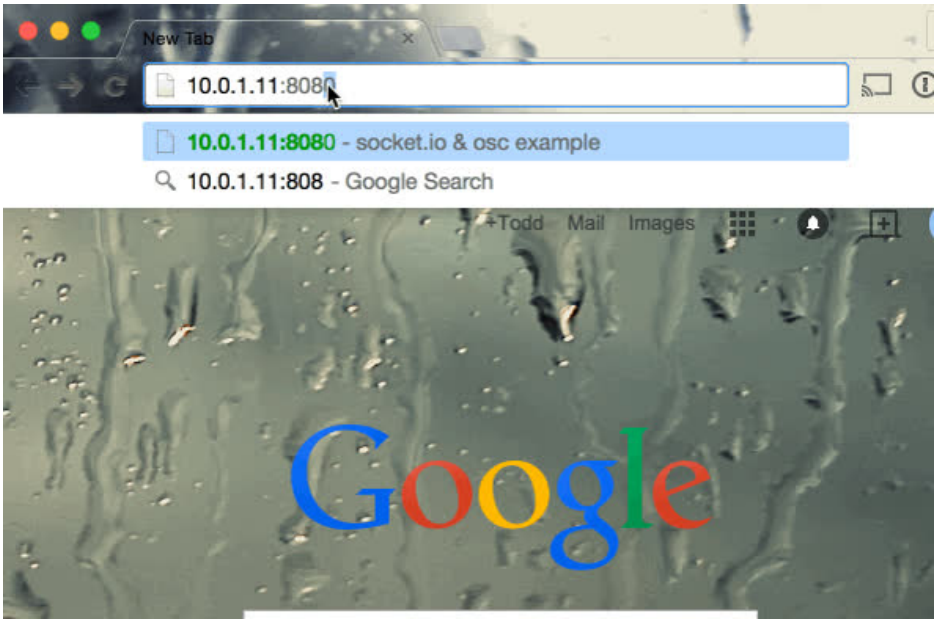
For this example we will be using the *osc* example patch in Max. The Pure Data and Chuck examples also will work, but to keep things simpler, we will be looking at this one environment.

### Starting the Node.js Script

To start the node script for this example, navigate to the osc-examples folder on your Pi and start the script by running **node socketio.js.** Once you have the script started, leave the window open.


### Opening the Test Web Page

Next, you will need to open the test web page in a web browser. You can see in the output of the node script that the HTTP server is listening on port 8080. To open the page, you will need to open the IP address of your Pi at port 8080 in the format of *http://x.x.x.x:8080*. Replace *x.x.x.x* with the IP address of your Pi.




## Open the Test Patch

As I mentioned earlier, I will be demonstrating this example using the Max *osc* example patch, but you can use the Pure Data or Chuck examples for the same results. Just as we did when testing communication, we will need to point the patch at the IP address of the Pi.

Once you have changed the IP, you can click the **/socketio** message button in Max to send a message to the browser! You will see the inputs update with the sent values, and the output printed to the page. You can also send values to Max by changing the inputs and clicking send. You will see the number boxes update in Max.


## Behind the Scenes

The Node.js script that is acting as the broker between the web browser and Max is fairly simple. The node script simply listens for OSC messages and passes them to socket.io, while at the same time listening for socket.io messages and passing them to Max via OSC. Here's a simplified snippet of the script that listens for OSC messages and passes them to socket.io.

```
var udp_server = dgram.createSocket('udp4', function(msg, rinfo) {

  // parse message
  msg = osc.fromBuffer(msg);

  // send args to browser
  io.emit('osc', {
    x: msg.args[0].value,
    y: msg.args[1].value
  });

});
```

Here's a simplified version of sending data from the browser to Max via OSC.

```
socket.on('browser', function(data) {

    var osc_msg = osc.toBuffer({
      oscType: 'message',
      address: '/socketio',
      args:[{
        type: 'integer',
        value: parseInt(data.x)
      },
      {
        type: 'integer',
        value: parseInt(data.y)
      }]
    });

   var port = 9999,
        ip_of_computer = '10.0.1.9';

    udp_server.send(osc_msg, 0, osc_msg.length, port, ip_of_computer);

  });
```

If you would like to look at the full source of the example, you can find it in the osc-examples GitHub repository (https://adafru.it/eUn).

## Sending Data from Hardware

For this example, we will be using a PiTFT send touch data from the Raspberry Pi. You could use this same method to send data from a wide range of sensors, but to keep things simpler, we will be focusing on the PiTFT touch data.

For this example you will need to have followed one of our PiTFT touchscreen installation guides, such as the guide for the 3.5" touchscreen (https://adafru.it/eno). Once you have successfully installed the PiTFT touchscreen, you will be ready to continue.

### Installing the Dependencies

First, run the following command to install the build dependencies.

```
sudo apt-get install evtest tslib libts-bin libts-dev libcairo2-dev
```

Then, **cd** into the *pitft* folder located in the *osc-examples* folder on your Pi and run **npm install** to install the node.js dependencies.

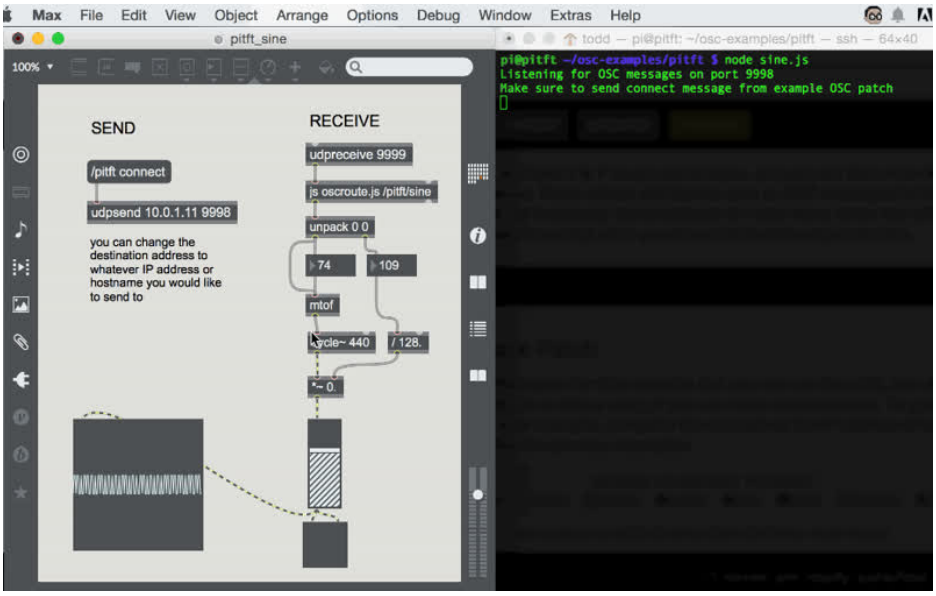

### Starting the Node.js Sine Script

The first script will take X & Y touch coordinates, and convert them into MIDI pitch and velocity values. These values will then be sent as OSC messages to Max, and used to control the frequency and amplitude of a sine wave. Enter the following command to start the script after you have installed the dependencies.

```
$ node sine.js
```

### Sine Wave Example

I will be using Max again for this example, but you can use the *pitft_sine* example in Pure Data or ChucK to follow along if you use those environments. To get started, open the *pitft_sine* example, and point the example at the IP address of your Pi as we have done for the previous examples. Press the connect **/pitft** connect button in Max to connect the patch to the Pi. If everything works as expected, your PiTFT will turn bright blue.



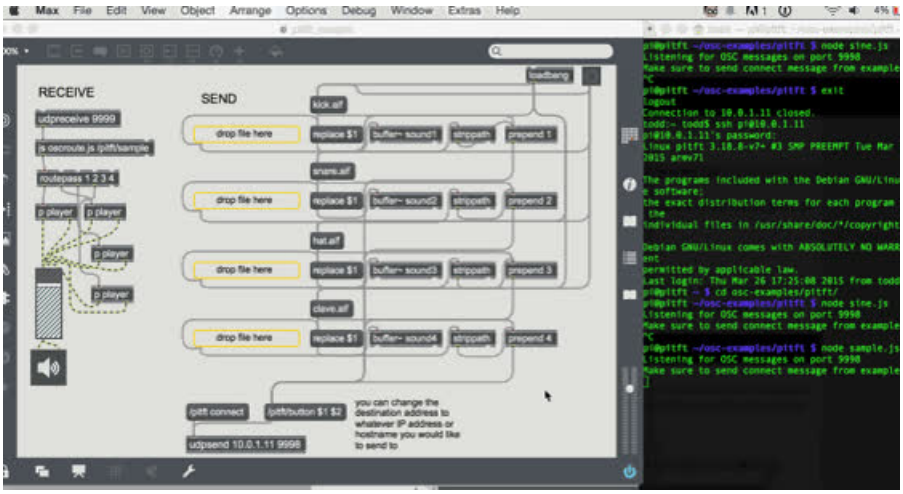

You can then drag your finger along the PiTFT to control the pitch and volume of the sine wave. Dragging your finger from left to right will change pitch, and dragging from top to bottom will control volume.

The source for the node script that transforms the touch data to OSC messages can be found in the osc-examples GitHub repository (https://adafru.it/eUp).

## Sample Trigger Example

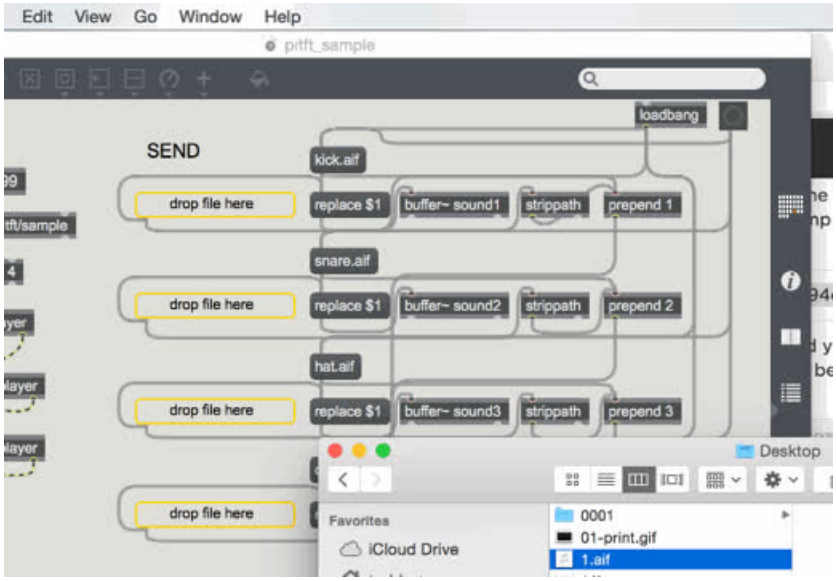
You can use the same process to start the sample trigger patch. First start the *sample.js* script on the Pi, which can be found in the *pitft* examples folder.

```
$ node sample.js
```




Then, open the *pitft_sample* example in Max, Pure Data or Chuck on your computer and replace the IP with the IP of your Pi as we have done in the previous examples. Then, click the button in the upper right to load the samples. You should see the names displayed on your PiTFT as shown in the video.

You can click on the sounds to trigger them in Max, and you can change the loaded clips by dragging new files into Max. The filenames will be automatically updated on the PiTFT.




The source for the node script that builds the UI and transforms the touch data to OSC messages can be found in the osc-examples GitHub repository (https://adafru.it/eUq).

## Modifying the Node Examples

If you need to modify the Node.js examples to fit your needs, your best resource will be the documentation for the osc-min (https://adafru.it/eUr) library.

## Feedback

Hopefully this guide has helped you get started with basic communication between a remote device and your favorite audio/visual programming environment. If you have any questions, please feel free to leave feedback. We'd love to hear your thoughts!