Extracted from:

Programming Sound with Pure Data

Make Your Apps Come Alive with Dynamic Audio

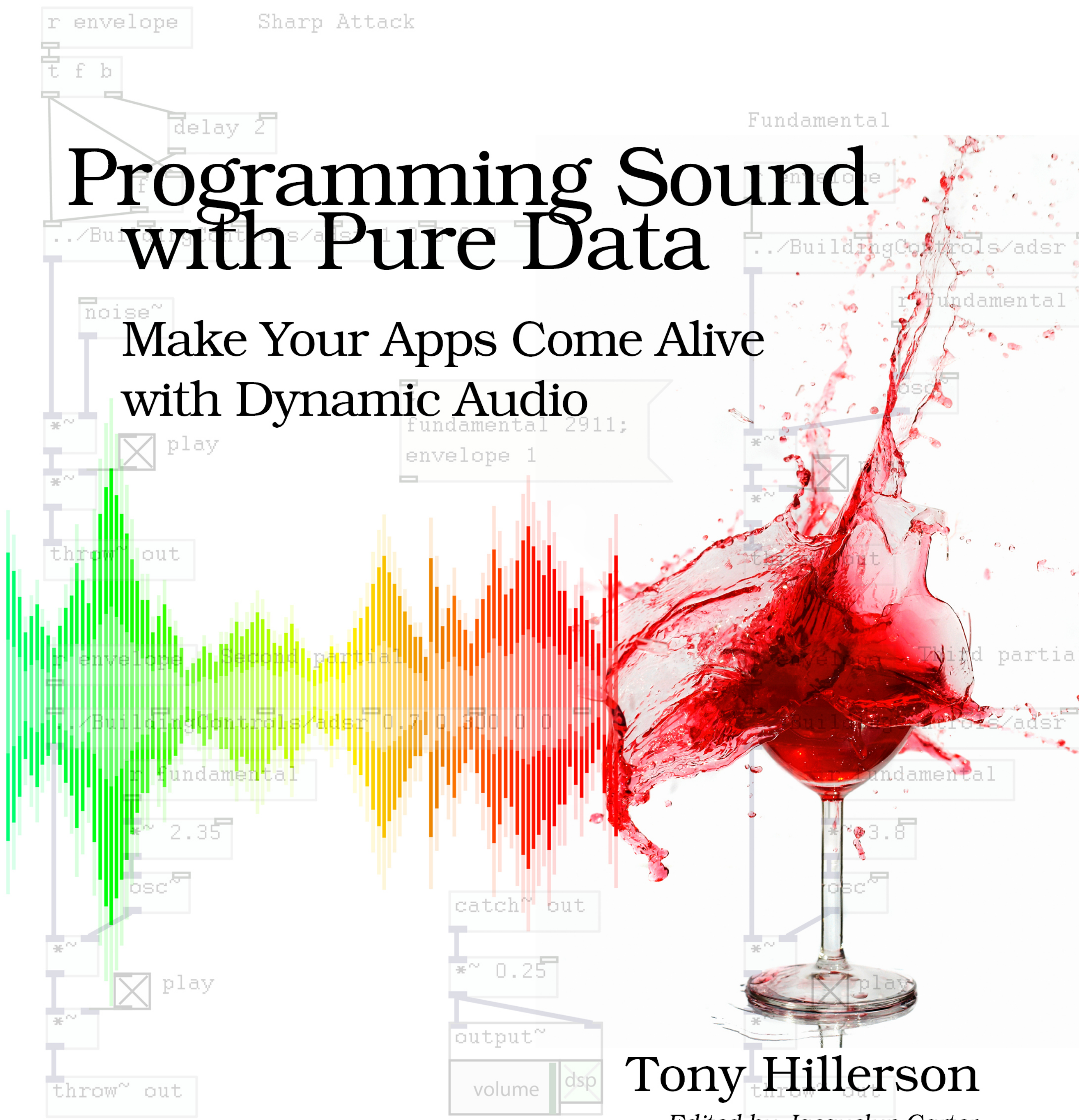
The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Programming Sound with Pure Data

Make Your Apps Come Alive with Dynamic Audio



Tony Hillerson

Edited by Jacquelyn Carter

Programming Sound with Pure Data

Make Your Apps Come Alive with Dynamic Audio

Tony Hillerson

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The team that produced this book includes:

Jacquelyn Carter (editor)
Potomac Indexing, LLC (indexer)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-93778-566-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January, 2014

Now let's get started programming Pd. In this chapter we'll cover a few important tools Pd has for generating sound, controlling it, and outputting the sound to the computer speakers. Most importantly, you'll learn how to make sound!

When you are done with this chapter, you will

- Understand some basic concepts about sound
- Connect things together to make a signal flow to the sound card
- Control the volume of sound we make
- Make sound!

Time to get hands-on with Pd.

Finding Your Way Around

We'll start with a new Pd file and discover a few things about editing. Open Pd, and you should see an image like Figure 1, *Editing a PD File,* on page 6.

In the top left are some meters to show input or output volume, in the top right is a check box marked *DSP*, and the rest of the window is taken up by a console with log messages. Notice that the DSP check box in the top right is unchecked. Leave it unchecked for now.

DSP stands for *digital signal processing*. This control toggles whether Pd processes and outputs sound to the operating system and on to your computer speakers. Always be thinking about what kind of sounds you're about to make and how to quickly make them stop. You can't easily replace your ears. If you are wearing headphones, check the system volume before you get ready to make sound with any application, including Pd. Also, remember that in Pd if whatever you do makes a sound you don't want to hear anymore, you can quickly jump to this main Pd window and uncheck DSP.

Creating a New Patch

Open a new Pd patch by using the File > New menu, or simply pressing ^N. A new window should open with a nice, clean blank *canvas*. Now open the Put menu and click Object. A dotted blue box is placed onto the canvas with a text field inside and ready for typing. If you move your mouse around the canvas, the object will follow it until you click or you press a key on the keyboard. Click on the canvas to place the object box in the top-left area of the canvas.

If You're on a Mac...

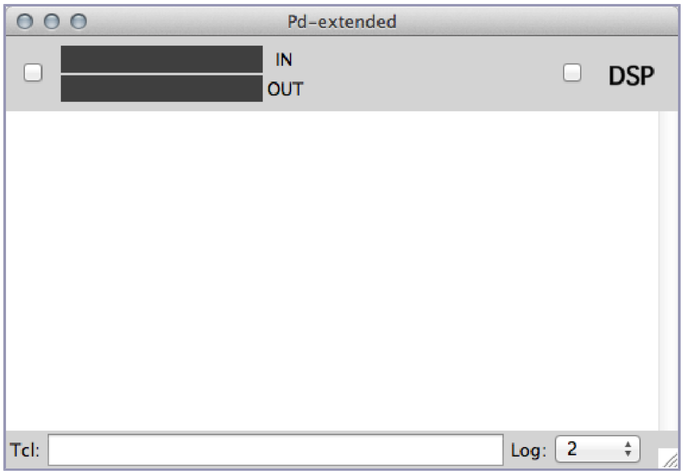


Figure 1—Editing a PD File

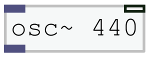
On the Mac you use Command (⌘) instead of Control (^) for any of the key commands you'll find in this book, but I'll give all key commands using the ^ key.

Now, if you click on the canvas away from the object, you'll notice that the dotted line around the box becomes red. This shows that this box doesn't mean anything to Pd right now. If you click back into the box, you're still able to edit it. You can also delete the box from your keyboard, but if you simply click the box, you'll find that the object is in edit mode and your delete key will delete characters inside the box instead of the box itself. The best approach is to band-select—click and drag your mouse around the object(s) you want to delete—and then delete it. Try that now: band-select the object and delete it. You should find yourself with an empty canvas again.

If you're going to use Pd for any length of time, you should get comfortable with a few simple key commands to put things on the canvas. The key sequence for putting an object on the canvas is ^1. Try this now: press ^1 and notice how an object is created wherever your mouse pointer is in the canvas. Once the object is on the canvas, you can drag it to position it where you want it.

Working with Objects

We now have an object on the screen. An object is in some ways much like an object in your favorite programming language: a building block that has some functionality. But, like in most programming languages, there's not a lot we can do with just an object; we need to tell Pd what kind of object we want. We do this by typing the object's *class name* in the box. This may be a little surprising if you've worked in a visual programming environment before, where you usually pick exactly the type of thing that you want from a set of menus, and then end up with complex property sheets to edit all sorts of parameters. Pd doesn't muck about with that sort of thing. It's visual, but it's simple. The following image shows the anatomy of an object.



Think of the class name of the object as something between a class, a function call, and a keyword in your favorite programming language. Another apt analogy is a shell command, such as `cd`, `grep`, or `sed`. The object box takes the name of the type of object you want to use and a list of optional arguments, just like a shell command, and just like a shell command, an object can vary widely in power and complexity. An object's outlets and inlets are the small bars at the top and bottom. Clicking and dragging from an inlet to an outlet will create a connection between the two. We'll look at connections and signal flow in just a second.

When typing in an object name, if you make a mistake about what you want, Pd will quietly tell you. Let's try it. Click inside the object you created with ^1 and type the word "nothing" into the box. Now click outside of the box. First of all you should see that the object's box has a red dashed line around it, just like the empty box did. Now switch back to the main Pd window and look in the console. At the bottom of the log messages you should see this:

```
 nothing
... couldn't create
```

Pd couldn't create an object called `nothing` because it doesn't have an object type named nothing. If you ever try something that doesn't work but you're not immediately sure what's wrong, check the log in the console to see if Pd has more information for you. Also, remember Pd's help, which contains a lot of information about the objects that do exist.

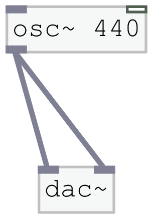
Now we've covered how to create an object with both the menu and the quicker key sequence, position the object on the canvas, and tell Pd what type of object we want. We haven't yet created an object that Pd understands, though, so let's make Pd actually *do something* by creating an object Pd does know about: an oscillator.

Hello Concert A

An *oscillator* is something that moves back and forth between two states, and in Pd it's an object that we can use to make sound. We'll get into more detail in a second, but for now let's create one and wire it up to the computer speakers.

Making an Oscillator

Create an object, place it near the top of the canvas, and type in `osc~ 440`. Now create another object underneath the first and type in the name `dac~`. Move your mouse over the outlet: the small gray box on the bottom left of the `osc~`. You should see your mouse pointer turn from the rather quaint pointing hand icon to a ring. Drag from that point to the left gray box on the top left of the `dac~` object, its left inlet. Now drag another line from the same bottom-left connection point on the `osc~` to the top-*right* box on the `dac~`, its right inlet. You should have a canvas that looks roughly like this figure.



Let's hear what it sounds like. Check your volume to make sure it's low—say, in the lower quarter of full volume—and go to the main Pd window. Click the DSP button on, and you should hear a high-pitched tone coming from your speakers. When you're done listening, toggle DSP off.

Signal Flow

We've just created two objects and made Pd process and produce some sound. First of all, notice that both object names have a tilde (~) at the end of them. By convention, object names that deal with *scalar values*, or numerical values that don't make sound directly, have no tilde, and objects that have a tilde at the end deal with *signals*. That means that the object either produces or modifies a signal, a numerical stream of data that will get turned into a sound.

To understand what this patch does, let's start with the second object on the canvas, `dac~`. A `dac~` sends a signal to the sound card. We'll dig in to how digital sound works a little later, but for now, know that DAC stands for *digital-to-analog converter*, and in this case it means the process by which your computer sound card makes sound. Whenever you hook a signal up to an object named dac~, you are sending a signal to your computer's sound card. The signal flows from an object's outlet into another object's inlet, and on until it reaches a dac~ and we're able to hear the signal as sound.

Now, back to the first object we created, the osc~. The object's name is 'osc~', which stands for *oscillator*. For now think of an osc~ as a sound-wave generator. If you're curious, the wave that it makes is a sine wave (a cosine wave, to be exact), but don't get too far ahead—we'll get into waves later.

As we saw, the little shaded boxes that are on the corners of the objects are the *outlets* and *inlets*. Inlets are at the top, and allow incoming data to control the object. Outlets are on the bottom, and send data out of the object. The lines that we drew from the outlet of the osc~ to the two inlets of the dac~ are called *connections*. Connections are fairly self-evident once you know you can draw them. You may notice a subtle difference between different connections: connections that carry signals instead of just numbers are a little thicker.

Getting Help

Pd comes with a nice help library that we can access and search from the Help menu. Even more useful, right-clicking an object will bring up a context menu that has an option for getting help about a particular object type. Try it now: right-click the osc~ object and read the help entry on the oscillator. Notice how it's broken into three sections—inlets, outlets, and arguments—and has links to related objects at the bottom. Whenever you want to know more about anything we cover in this book, the context help is a great place to start.

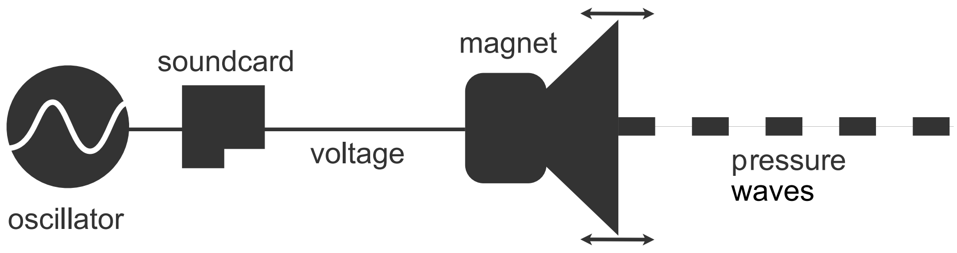
The Argument to the osc~ Object: Frequency

We passed an argument or parameter to the osc~: 440. Think of this like the object's constructor: we're creating an osc~ with an initialization value of 440. The value is optional, but by doing this we've created an oscillator with a *frequency* of 440 hertz. Abbreviated Hz, the unit hertz measures the number of times something that cycles between two positions does so in one second.

To illustrate frequency, think of a swing on a playground with a kid going full speed back and forth. The swing is an oscillator. If the kid's legs start out pointing forward, swing back as far as they can go, and then return to the point where they started all in the space of one second, then the frequency of the swing is 1 Hz. We also say that the swing has a *period* of one second. Period is the inverse of frequency.

An Audible "Hello World"

Now back to our osc~. It's oscillating 440 times per second. What does that mean? What's doing the oscillating? The wave that osc~ generates is oscillating at 440 Hz; to be more accurate, the signal the osc~ generates describes a wave with that frequency. The figure here can help you visualize this process.



When the signal gets sent to the sound card and converted to something that can move your computer speaker or headphones, at that point the speaker is vibrating at 440 Hz. That speaker is attached to a magnet that pushes and pulls it back and forth, pushing air back and forth at 440 Hz, creating waves of high and low pressure in the air. Finally, parts of your ear are moving back and forth at 440 Hz. This is a simple description of how electronic sound works.

Now, why did I pick 440 Hz? It is a standard note in (most) Western music, called "Concert A," and often it's the note used to tune up an orchestra. In other words, you've just said an audible "Hello World" using Pd. Well done!

Things to Think About

Why are there two inlets to the `dac~`? See if you can guess, then turn on DSP and delete one connection by clicking and pressing delete. Listen to the difference. What do you notice?

What happens when you change the frequency of the `osc~`? How low can you go before you can't hear it anymore? How high? Hint: You may have to get up into the tens of thousands.