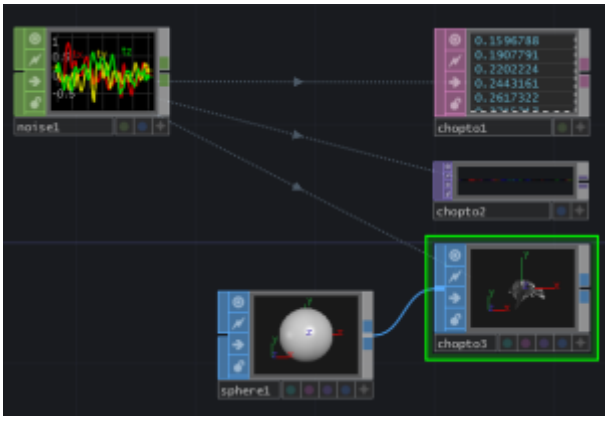
Matthew Ragan

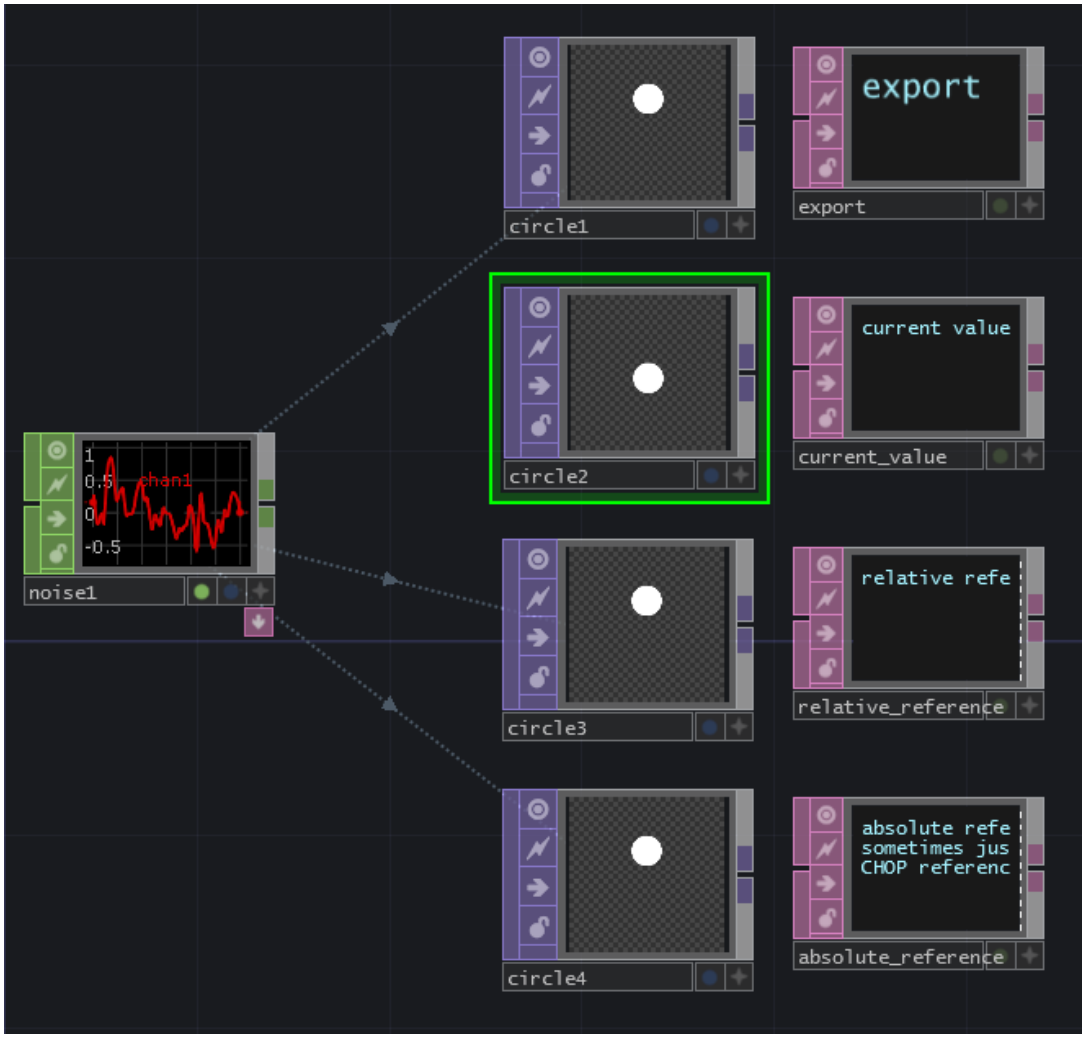
Media Maker and Interactive System Designer

Understanding Referencing | TouchDesigner



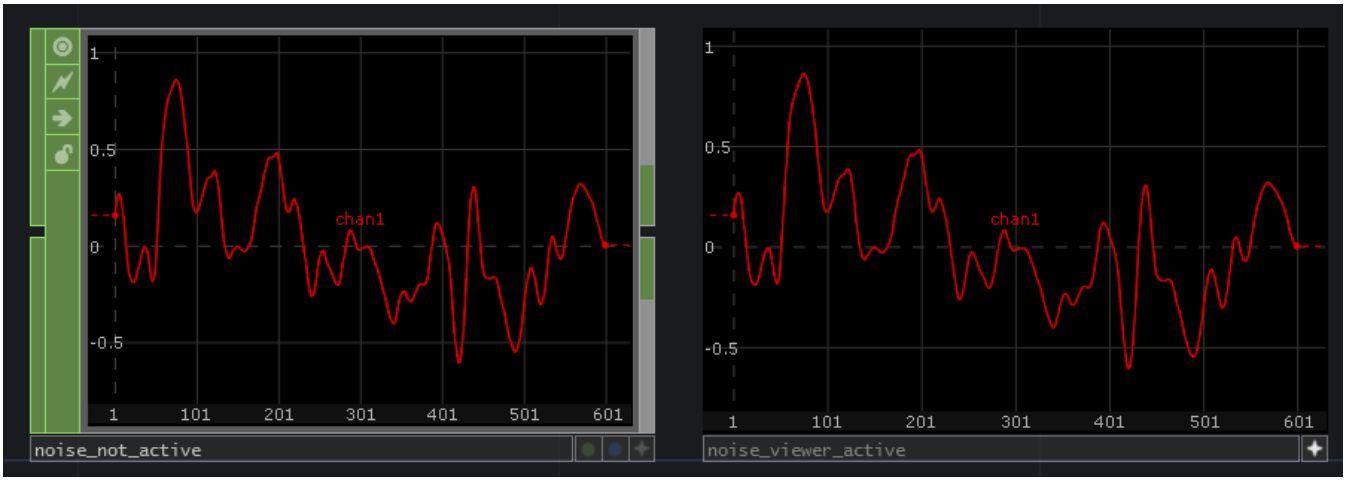
Referencing is one of the most powerful tools at the programmer's disposal in TouchDesigner. Referencing creates a direct link between two or more floats or integers. This allows you to link operators that are outside of their respective families – normally you can only connect CHOPs to CHOPs and TOPs to TOPs, but referencing allows you to create connections between nearly any operators. There are a number of way to create these links with references or expressions. In many of the other posts that I've written I often write about using expressions and references, but haven't taken much time to talk in depth about what this is, how how it all works. Let's change that.

I started thinking about this when I saw a post on the Derivative forum from a new user struggling with understanding what I'd written in some earlier tutorials. Expressions are something that I continue to learn more about, and open up all sorts of opportunities for faster, more streamlined, and more elegant programming. Let's start by looking at the typical kinds of referencing that you might do on any project. Specifically, let's look at how we might connect one family of operators to another. In this example we'll look at connecting a CHOP to a TOP, and all of the different ways we might do that.



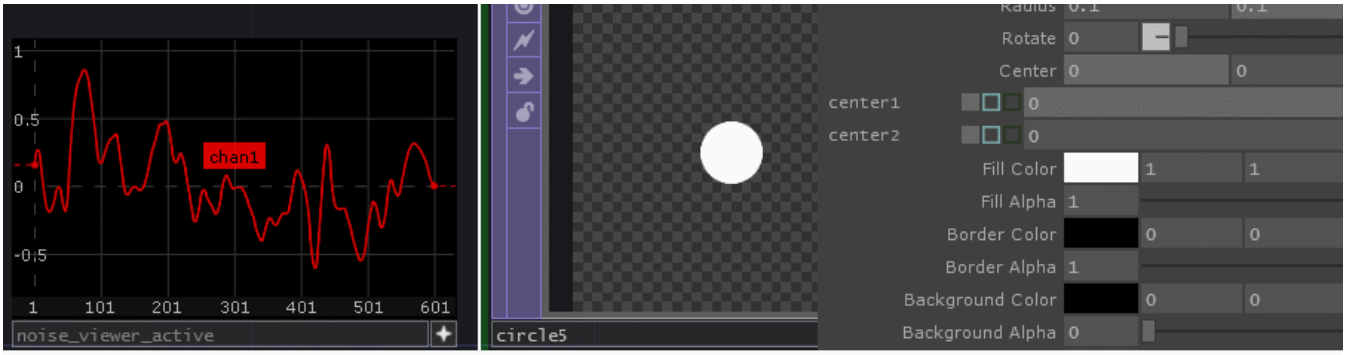
In the image above we actually are referencing the same CHOP in four different ways. We can start this by first talking about how we connect two operators from different families. In this example I'm going to use a noise CHOP and a circle TOP. I want to use the sudo random noise from the noise CHOP to drive the vertical position of my circle in my circle TOP. There are a two major ways that we can make this connection: dragging and dropping the CHOP onto the TOP, or writing an expression that connects the two of them. I often opt for writing the expression – I do this because I think it's good practice, and has helped me better understand the syntax and structure of using expression in references. We'll take a look at both of these methods.

Let's start with the drag and drop method. To use the drag and drop method we need the source operator to be viewer active (there are some exceptions, but it's a good rule of thumb that your source probably needs to be viewer active to do this). We can make an operator viewer active by clicking on the + symbol in the bottom right corner, or by holding down the alt key (holding alt will make all operators viewer active). You can tell an operator is viewer active because the color coded border disappears, and usually a portion of it is highlighted when you mouse over the operator.

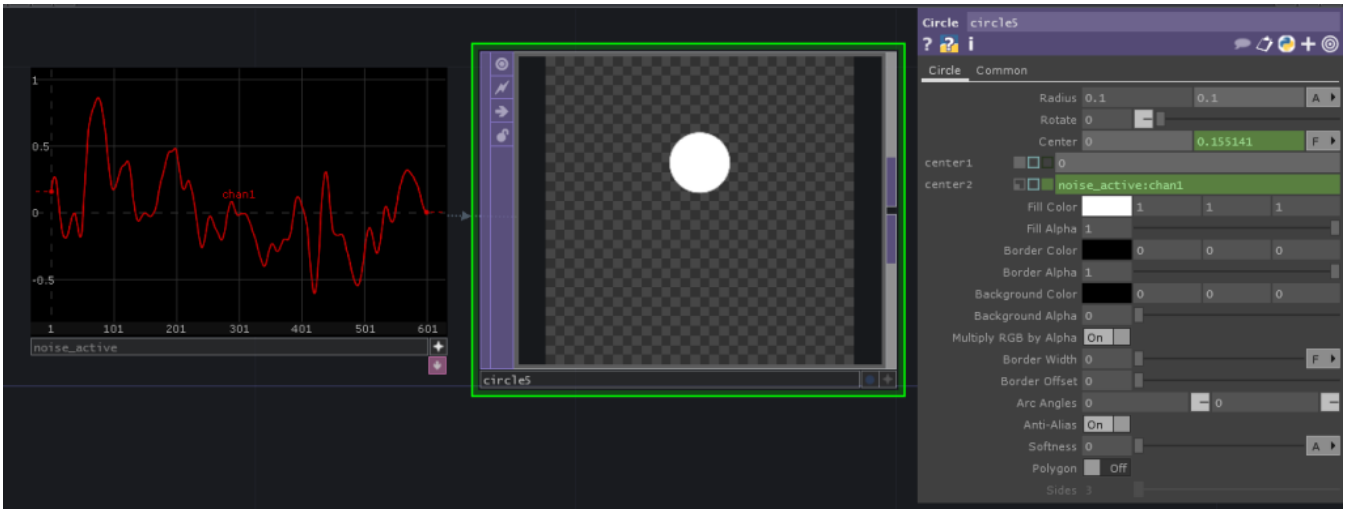


Let's take a moment to better understand the anatomy of an operator while we have this example handy. On the left upper corner of the operator we have a few different toggle switches – viewer, clone immune, bypass, and lock. Along the very bottom of our operator we have it's name (you can make changes to this field), any flags associated with the operator family, and the viewer active toggle. Having a solid sense of the anatomy of your operators becomes increasingly important the longer you work with TouchDesigner.

Alright, now that we know how to toggle our viewer active mode on and off, and know a little more about our operators' anatomy lets look at how to build a reference. Let's make a noise CHOP in our network as well as a Circle TOP. With the Noise CHOP viewer active, click on the name "chan1" in the viewer, and drag it to the Y parameter of the Circle TOP (I've made my circle a little smaller, to make this easier to see):



As you do this you should see a drop down menu appear, let's select "Export CHOP" from the list. You should now see the Y position (or center 2) changed to a green color. You should also see some text that shows up as well. Here's a closer look at just the paramater we've changed in the circle TOP:

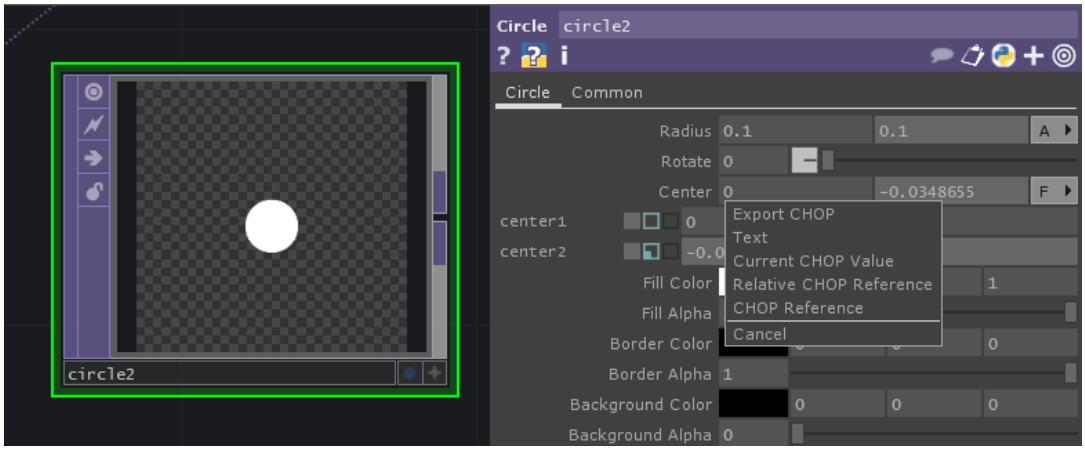


Looking closer we can see that the text reads: noise_active:chan1. Great, but what does that mean?! Well, if we take a closer look at our Noise CHOP we can see that I changed the name of that operator to "noise_active" – we also see that the name of our noise channel is "chan1". If we were to abstract what we're seeing in the export language we might write something like this:

source_operator_name:source_channel_name

Exporting is a fine way to connect operators, but it's not my personal favorite. I say this because exporting creates a locked relationship. Once you've done this you can't change the text in the target operator. Exporting creates a much more permanent relationship between your operators. To remove the export you'll need to right click on the parameter filed and select "remove export." Surely there's a better way to connect operators?!

In fact, there are still three more ways to connect operators. Taking a closer look at the drop down menu that appears when we use the drag and drop method we see the following that there are several Methods:



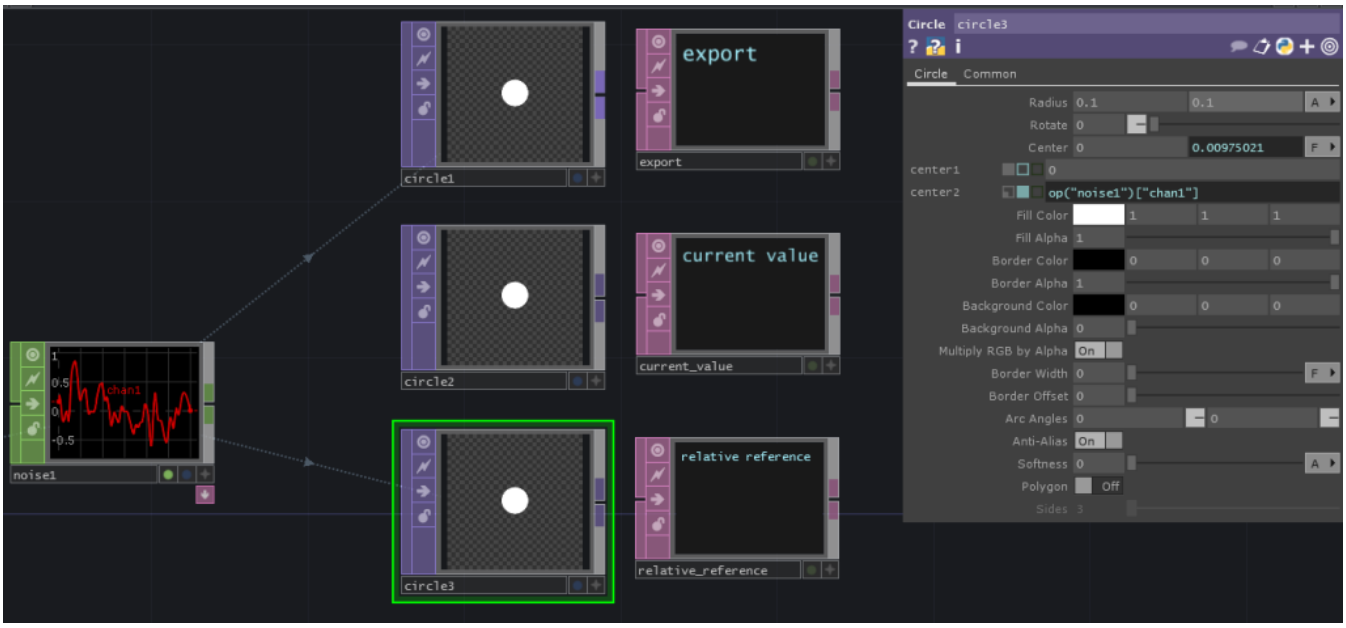
All of these are ways that we can connect two operators together, and many of them yield the same results, so what gives:

- **Export CHOP** – we've already seen this method, and we know that one of its limitations is that it creates a fixed relationship between two operators. This is excellent for creating something more finished for locked in nature.
- **Text** – Text exports the pathway to a particular channel.
- **Current CHOP Value** – this exports the value of the operator in question at the precise moment that you drag and drop. Rather than a continually updating value this is just a single float or integer.
- **Relative CHOP Reference** – the relative reference exports a python expression that points to the operator being referenced. A relative makes for easy cutting and pasting so long as the network hierarchy relationships remain constant between operators.
- **CHOP Reference (sometimes called Absolute Reference)** – the absolute reference also creates a python expression pointing to an operator. The difference here is that it includes the precise pathway to the operator in question making cutting and pasting a bit more frustrating.

For now we'll take a pass on the "Text" and "Current CHOP Value" options as these have more limited uses. Let's now take a close look at our Relative and Absolute Reference options.

Relative Referencing

Let's go ahead and make another circle in our network, and this time let's create a relative reference between our noise CHOP and our circle TOP.

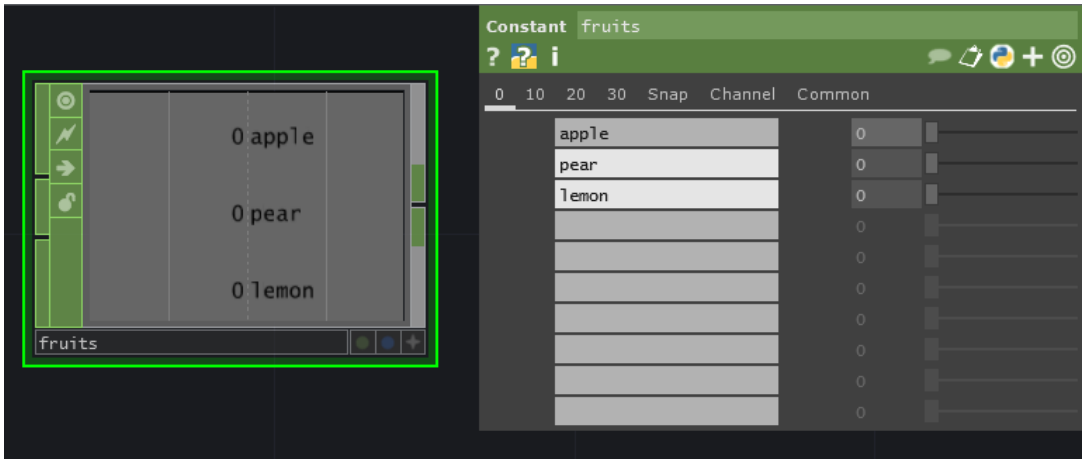


Taking a closer look at our expression we can see that it reads:

op("noise1")["chan1"]

Okay, what does this mean? Lets start by looking at the syntax of this expression. First we can see that we're looking for an operator. We know this because our expression starts with **op()**. Next comes the name of the operator in quotation marks. As an important note, Python doesn't care if you use double quotes or single quotes so long as they match. This means that ***"noise1"*** and ***'noise1'*** are both equal and produce the same results; ***'noise1"*** or ***"noise1'*** however will not work. Finally we see the name of the channel in question in brackets and in quotes – **["chan1"]**. This means that our syntax looks something like operator("exact_name_of_operator")["desired_channel"]. Okay, let's look at another example to make sure we have a firm understanding of how relative referencing works.

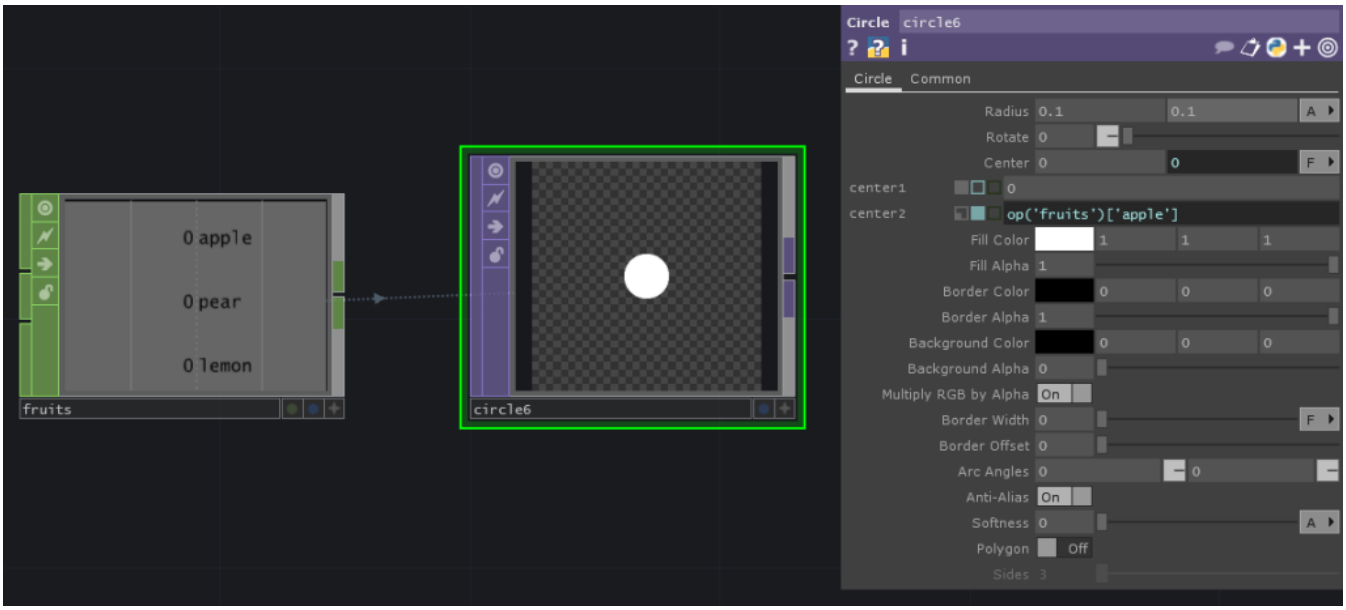
Let's make a Constant CHOP. Let's name change the name of the constant to "fruits" and name the first three channels, "apple" "pear" and "lemon". You should have something that looks like this:



Alright, now lets add a circle TOP to our network. This time, instead of using the drag and drop method we'll write out the Python expression to create a reference to our constant. We'll start by referencing our apple channel. This means our expression is going to be:

op('fruits')['apple']

You can write this directly in the expression directly in the target parameter field for the target operator. When you're done you should have something like this:



If you drag the slider in the constant CHOP to the right, you should now see the circle move up in the viewer. So we've successfully connected our circle TOP to the apple channel, why is this any better than just exporting? Well, let's say that for whatever reason you change your mind while you're programming and decide that instead you'd prefer for the circle TOP to be connected to the "pear" channel? Written as an expression we can make that change simply by deleting "apple" and replacing it with "pear" or "lemon". Our expressions then would be:

op('fruits')['pear']
op('fruits')['lemon']

Additionally, if we've written a reference as an expression we write some math directly into our reference. We might, for example, only want half of the value coming out of the apple channel. In this case we'd write the expression:

op('fruits')['apple'] * 0.5

This would divide ever value in half, changing our scaling from 0 – 1 to 0 – 0.5 instead. We can also use this method to multiply a channel by another channel. For example maybe we want to create a relationship between two different channels from our fruits constant. We might write the expression:

op('fruits')['apple'] * op('fruits')['lemon']

You could just as easily do this with a Math CHOP, but you might find that just writing the expression is faster, simpler, or more tidy.

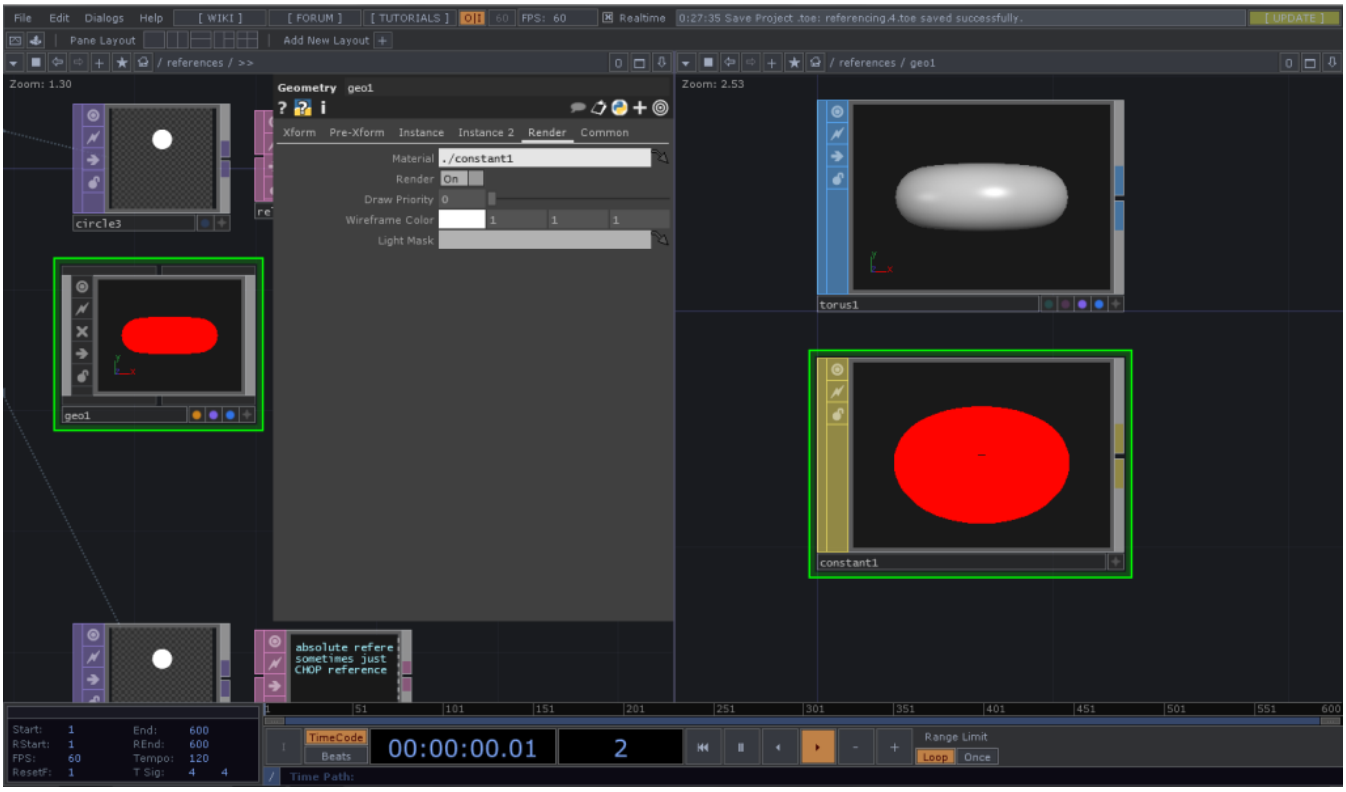
Before we move on, there are two more modifiers that we need to know when writing relative references:

./
../

What on earth are these all about? Well, these are handy directory pointers. At some point you will surely end up wanting to reference an operator that is another part of your network – a control panel, a material, a slider, you name it – if you program in Touch long enough, you're gonna need these. So what do they mean:

./ – this modifier means the network inside of me
../ – this modifier means in the network above me

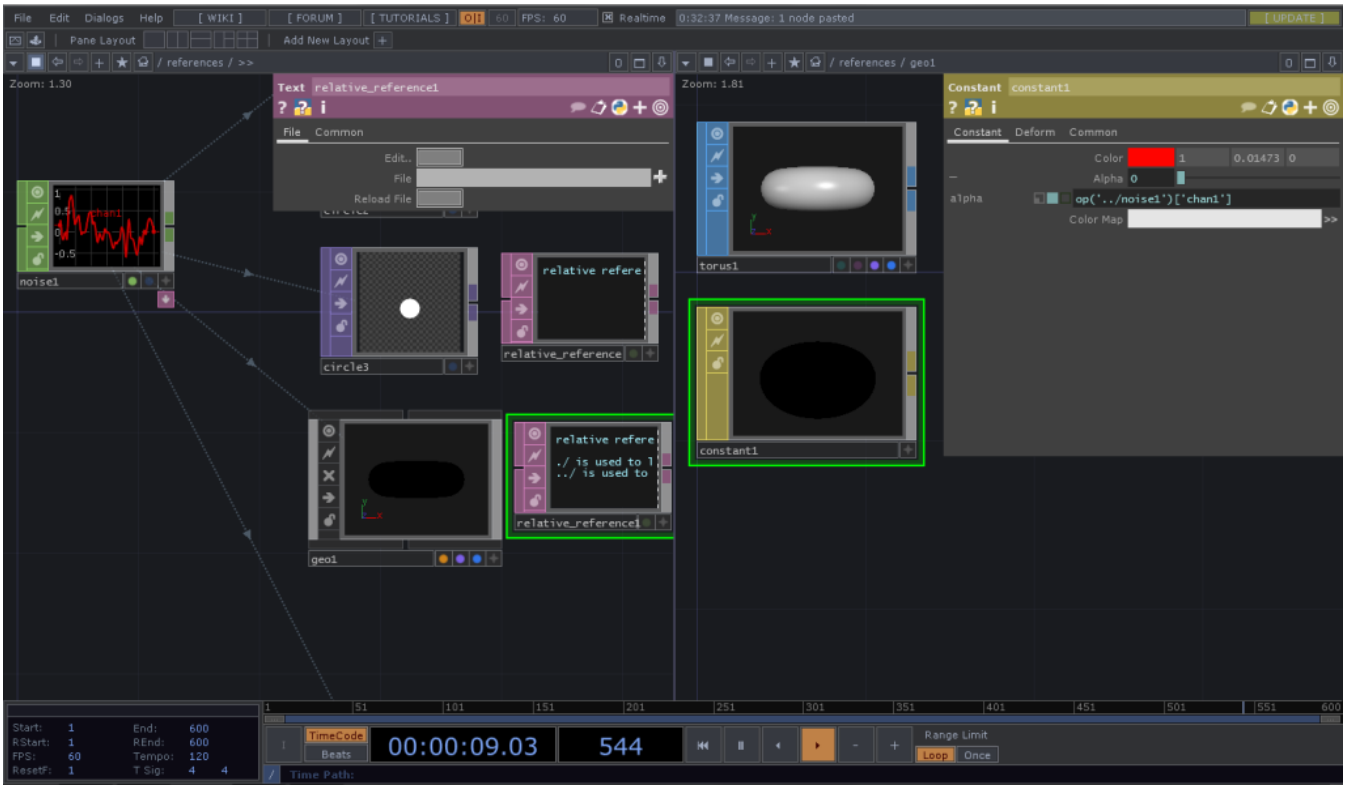
If you're scratching your head, that's okay. Let's look at an example. Let's say that we have a Geometry component, and inside of it we have placed a material – a constant that's red. A relative reference for that material would be "./constant1". This means, look inside of me for the material called "constant1".



So how does ../ work then? Imagine that you'd like the alpha of the constant in our geo to be connected to the noise in our parent network. Here we can write a reference that looks like this:

op('../noise1')['chan1']

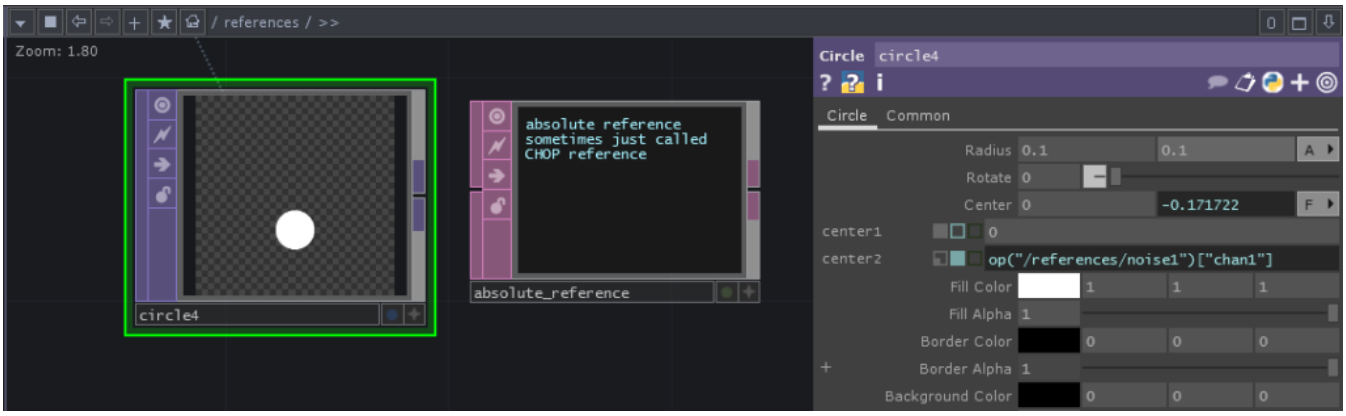
Here's what that would look like:



Absolute References

Now that we understand what relative references are, what are absolute References? Unlike relative references, absolute references require the entire network path to an operator. In the case of our noise and circle example, that means that our reference looks like this:

op("references/noise1")["chan1"]



Absolute references require that you know exactly where in your network you're referencing an operator from, because you have to use the entire network path. That sounds like a pain, so why use them? Well, let's imagine that you're building a complex program and you're trying to be as tidy and organized as possible. You might build a large change of your user interface in a single location. This means that all of the sliders, buttons, and menus that are being called all live in the same container. In this case, using an absolute reference makes good programming sense. Relative references will leave you constantly trying to figure out how many ../ to use when referencing your user interface. Absolute calls don't require this, as they point to a very specific place in the network. You can even simplify this by making sure that all of your buttons and sliders are joined with a merge CHOP.

To get a better sense of how this works, download the example .toe file and look at the last example that's driven by sliders that control the level TOP. There's lots more to learn about expressions, but practicing your references will help you begin to understand the syntax and logic of how they work.

Download this example toe to learn and explore some more – referencing toe

This entry was posted in How-To, media design, Programming, Software, TouchDesigner and tagged Arizona State University, ASU, Derivative, Grad School, graduate school, media design, programming, TouchDesigner, TouchDesigner Tutorial on June 1, 2014 [https://matthewragan.com/2014/06/01/understanding-referencing-touchdesigner/] .

2 thoughts on “Understanding Referencing | TouchDesigner”

Pingback: Understanding Referencing Part II | TouchDesigner | Matthew Ragan

Pingback: Python in TouchDesigner | Writing Python References | TouchDesigner | Matthew Ragan

Comments are closed.