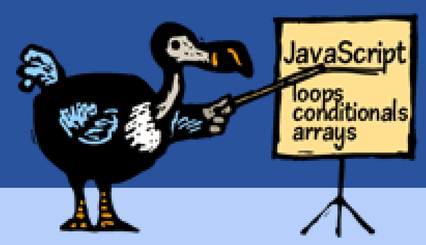
MotionScript.com

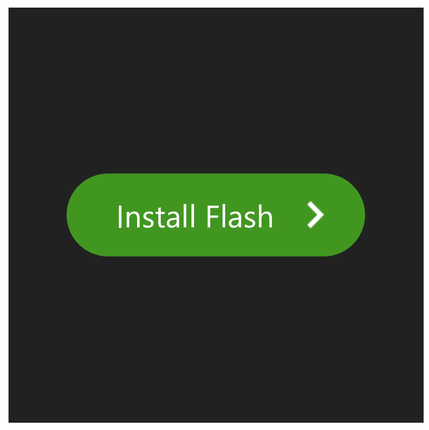
Expressioneer's Design Guide



Looping wiggle()

Idea

Without question, **wiggle()** is an extremely handy and versatile tool for adding an element of randomness to a project. Sometimes though, it would be extremely useful to be able to get the wiggle motion to loop. Here we'll take a look at a way to accomplish this by using a bit of clever math and taking advantage of one of **wiggle()**'s seldom used parameters.



With a little math wizardry, we can get wiggle() to loop seamlessly.

Design

Let's say we want to create a three-second loop of wiggle motion. Visualize a graph showing the path that our layer traces out as it moves randomly from time zero through three seconds. If you examine the lower movie to the right you'll see such a path traced out by the blue dot. It starts in the lower left corner and ends up near the middle of the right edge after three seconds. Then it repeats.

Now examine the path traced out by the red dot. Notice that it starts near the right edge but ends up in the lower left corner - exactly where the blue dot starts. That seems like quite a coincidence, right? Well, there's a little trickery going on here that is key to making this all work. It turns out that the red and blue dots are actually tracing out the same random wiggle path. The trick is that the red dot is tracing out the three-second portion of the path that occurs before time zero.

Let that sink in for a second. The red dot is tracing out the portion of the path that occurs between minus three seconds and zero. The blue dot is tracing out the portion that occurs between zero and positive three seconds. We are able to access these negative time values of **wiggle()** by using the seldom-used fifth parameter of **wiggle()** - time. This parameter works very much like the valueAtTime() method you can use to retrieve a property's value at any given time. Whatever time you specify, you get the **wiggle()** value that would occur at that time. By putting a negative value in for time we can get a layer to trace out a portion of the path that occurs before time zero.

OK - that's great but how does that help us? Well, all we have to do now is blend the two portions of the path together so that our layer starts out where the blue dot starts, but ends up where the red dot ends up (which happens to put it back where it started, thus creating a loop). This is exactly how the seamless loop of the white dot was created. This blending of the paths turns out to be quite easy to do using the extremely useful **linear()** interpolation method.

The looping path of the white object is created by blending the paths of the blue and red objects.

Copy and Paste Code

```
freq = 1;
amp = 110;
loopTime = 3;
t = time % loopTime;
wiggle1 = wiggle(freq, amp, 1, 0.5, t);
wiggle2 = wiggle(freq, amp, 1, 0.5, t - loopTime);
linear(t, 0,  loopTime, wiggle1, wiggle2)
```

In its fully populated form, **wiggle()** can take up to five parameters. We only need to use the first, second and fifth parameters for this task. However, to get to the fifth parameter, we have to specify the third and fourth paramters. So we'll just plug in the default values (the values that After Effects uses if only two parameters are specified).

Analysis

Create variable "freq" to represent wiggle frequency and set it to one wiggle per second.

Create varaible "amp" to represent wiggle amplitude and set it to 110 pixels.

The variable "loopTime" will represent the length of our wiggle loop (3 seconds).

Varaible "t" represents how far we are into the current loop cycle. By using the JavaScript modulus operator (%) we get a time value that resets to zero when it reaches 3 seconds.

```
1 freq = 1;
2 amp = 110;
3 loopTime = 3;
4 t = time % loopTime;
5 wiggle1 = wiggle(freq, amp, 1, 0.5, t);
6 wiggle2 = wiggle(freq, amp, 1, 0.5, t - loopTime);
7 linear(t, 0,  loopTime, wiggle1, wiggle2)
```

"wiggle1" is a variable that we use to capture the wiggle value that occurs between zero and 3 seconds. Note that we have specified the default values for wiggle()'s 3rd and 4th parameters so that we can get to the 5th parameter (time) where we plug in our time variable, "t".

Finally, we blend the two wiggles using the linear() interpolation method. As "t" goes from 0 to 3, the output of linear() starts at the value of "wiggle1" and ends at the value of "wiggle2".

Notice that variable "wiggle2" is calculated the same way as "wiggle1" except for the time parameter. Here we're using a time that runs from -3 to zero.