TONES Parameter

anematode

January 5, 2019

1 Who is this intended for?

It's intended for me. I might read this in a few months because I didn't know what the hell I was doing. Thus, it's written in a somewhat understandable way, but it won't try to explain ideas that aren't a product of this project.

2 Motivation

A parameter is an abstraction of a **value**. This value can have a multitude of *units*, which are supported as a helpful annotation, but should not change how things are calculated (exception: they may determine the parameter bounds, which will be discussed soon). Most units fall in the following categories:

gain: describe how much a signal should be amplified as a linear increase of *amplitude*, or the loudness of a signal. The most obvious used domain for this is 0 to ∞, but when described as a signal multiplication, it makes mathematical sense to extend its domain to $-\infty$ to ∞ where negative values simply cause a signal inversion.

Example: the signal gain of a few volume knobs, volume of a sustain (also could be in dB), many internal transformations

dB: describe how much a signal should be amplified as a linear increase of *amplitude*, or the loudness of a signal. This is more often seen, and is a logarithmic transformation of gain that more closely matches how our ears perceive loudness. The natural domain is $-\infty$ to ∞.

Example: the signal gain of most volume knobs, volume of a sustain (also could be in gain), a few internal transformations

frequency: describe how fast something cycles, often in hertz but potentially in a unit like beats per minute; however, even if it's in a different unit, the transformation from that unit of frequency to the equivalent value in hertz will be linear. The natural domain is $(0, \infty)$. When the frequency is 0, then the wavelength is $1/0$; we could make this ∞ as if we were working on the extended number line but I'd rather not.

Example: **tempo**, pitch

time: describe how long it takes something to occur; for the special case of wavelengths, this is just $1/f$ where f is the frequency. Natural domain is $(0, \infty)$ as well.

Example: $\underbrace{\text{delay time, LFO length}}_{\text{wavelengths}}$, attack length, release length, decay time for reverb

scale: a number from 0 to 1

Example: effect volume, potentially discrete "on/off" things

none: dimensionless values. Intervals are actually dimensionless values: for example, a detune of 1 cents up can just be thought of as multiplying by $2^{1/1200}$.

Example: detune, miscellaneous, most discrete values

Parameters have a large variety of modes, because they need to be dynamic yet very fast. I will refer to their parameters as properties for syntactical clarity henceforth.

Parameter properties fixed at creation time:

isAudio: true/false, does this parameter directly determine the behavior of an actual underlying web audio node/parameter

true examples: almost all volume nodes, frequency false examples: envelope automations, some reverb times depending on the algorithm

isStatic: true/false, can this parameter change value or is it fixed at creation time?

true examples: some internal transformations, whatever you don't want to change false examples: all modulatable parameters, customizable parameters that aren't modulatable

isModulatable: true/false, can this parameter have *automations* applied to it?

true examples: parameters that can have automation on them like volume, attack length false examples: static parameters or non-automatable parameters like some reverb length

isBeatTime: true/false, is this parameter describing a changing value in beat time or in playback time?

true examples: parameters controlling channel volume, pan false examples: parameters controlling an attack

The reason these should be fixed at object creation is because they have some degree of overhead if true/false (more overhead for isStatic if it is false, true for others). Nevertheless, they all support a couple basic commands, so it was easier to just put them in the same class for reasons that will be apparent later.

Parameter behaviors with each combination of *isAudio*, *isStatic*, *isModulatable*, *isBeatTime*:

If any are not true/false: throw an error at creation time

isA	isS	isM	isBT	Class	Behavior
N/A	true	true	N/A	N/A	invalid configuration

true	false	true	true	A	A parameter that has an editable automation timeline *in beat time* and the associated system to calculate the beat time → playback time → context time transformation, supports valueAt, derivativeAt etc. functions taking in beat time, and can have changes to its underlying audio parameter (given at construction time) scheduled by asking it to schedule automations between two beat times and which loop cycle it is on. Furthermore, a setValue function can be called that will set the underlying audio value to this value, but the automation will not be changed and the value will be overridden by any scheduled automation.
true	false	true	false	B	A parameter that has an editable automation timeline *in playback time*, supports valueAt, derivativeAt etc. functions taking in playback time, and can have changes to its underlying audio parameter (given at construction time) scheduled by asking it to schedule automations between two playback times. Furthermore, a setValue function can be called that will set the underlying audio value to this value, but the automation will not be changed and the value will be overridden by any scheduled automation.
false	false	true	true	C	A parameter that has an editable automation timeline *in beat time* and supports valueAt, derivativeAt etc. functions taking in beat time. Furthermore, a setValue function can be called that will set the underlying audio value to this value, but the automation will not be changed and the value will be overridden by any scheduled automation.

false	false	true	false	D	A parameter that has an editable automation timeline *in playback time* and supports supports valueAt, derivativeAt etc. functions taking in playback time. Furthermore, a setValue function can be called that will set the underlying audio value to this value, but the automation will not be changed and the value will be overridden by any scheduled automation.
true	false	false	N/A	E	A non-static but non-automatable parameter that supports a setValue function which changes the value of its underlying audio parameter given at construction time.
false	false	false	N/A	F	A non-static but non-automatable parameter that simply allows you to store a value used by other things. Supports the valueAt, derivativeAt, etc. functions.
true	true	false	N/A	G	A static, non-automatable parameter whose value is fixed at construction time and sets the value of the underlying audio parameter to this static value. It supports the valueAt, derivativeAt, etc. functions. It need not set it over and over assuming something else has changed it; it should assume that it is the only one controlling this parameter and thus set its static value at construction time.
false	true	false	N/A	H	A static, non-automatable parameter whose value is fixed at construction time and can be thought of as simply a constant. Again, it supports the valueAt, derivativeAt, etc. functions.

A parameter is constructed rather simply with the following options in the constructor properties object. Bolded properties are required, italic properties are required only under the given conditions, and plain text properties are optional for all.

class: A string "A" to "H" for which parameter class it is. *audioParam*: required only if *isAudio*, the web audio parameter that the parameter describes. *system*: required only if *isBeatTime*, the system which is referenced for beat time to playback time conversion. *value*: required only if *isStatic*: value at

construction; if omitted in non-required cases, does not change the underlying audio value and sets its own value to 0.