The Formal Definition of the *Erie* Grammar

This is a supplementary document for CHI 2024 paper, *Erie: A Declarative Grammar for Data Sonification*.
(Paper DOI: https://doi.org/10.1145/3613904.3642442)

1. Top-level Specification

```
Spec := {Title, Description, Auditory description
         (Stream | Overlay | Sequence), Design definition
         Dataset, Tick, Synth, Wave, Sampling, Config}
         For stream composition and customization
```

2. Stream: a unit sonification design

```
Stream   := {Data, Transform, Tone, Encoding, Config}
Overlay  := [Stream]  Playing multiple streams together
Sequence := [Stream | Overlay]  Playing one stream after another
                                (Note: a nested sequence = a sequence)
```

3. Data and datasets: what to be sonified

```
Data    := Name<String> | Url<UrlString> | Values<Array>
                  Refers to
Dataset := [{Name<String>, (Url<UrlString> | Values<Array>)}]
```
Allows for registering datasets and using them in overlaid or sequenced streams

4. Transform: modifying data for sonification design purposes

```
Transform := [Aggregate | Bin | Density | Fold |
              Calculate | Filter | ... ]
```

5. Tone: overall audio quality ≈ mark or glyph

```
Tone     := {ToneType,
             Continued<Boolean>,   When an audio property changes
             Filter}               Discrete: momentarily pause and resume
                                   Continuous: no pause and resume
                Oscillators
ToneType := 'default' | 'sawtooth' | 'triangle' | 'square'
    Musical | 'piano' | 'pianoElec' | 'violin'| 'guitar' | 'metal'
     Drums | 'hithat' | 'snare' | 'highKick' | 'lowKick' | 'clap'
     Noise | 'whiteNoise' | 'pinkNoise' | 'brownNoise'
           | *<String>  Custom instruments
Filter   := [FilterName<String>]
```

6. Encoding: mapping from data to audio

```
Encoding   := [Channel: ChannelDef]
Channel    := 'time' | 'time2' | 'duration' | 'tapSpeed'
            | 'tapCount' | 'pitch' | 'detune' | 'pan'
            | 'loudness' | 'timbre' | 'postReverb'
            | 'modulationIndex' | 'harmonicity' | 'speechBefore'
            | 'speechAfter' | 'repeat' | *<String>
                                  Custom channels via audio filters
ChannelDef := { ( {Field<string|[string][Channel='repeat']>,
   Dynamic channel      EncType, Scale}
Conditioned or static  | {Condition, Value<Any>} ),   For nested repetition
                 (Ramp)[Tone.Continued=True],
                 Aggregate, Bin, Inline data transforms
   For specific  (TimeUnit, TimeLevel)[EncType='temporal'],
   encoding types (Speech<Boolean>, By)[Channel='repeat'],
   and channels   (Tick<TickItem|String>)[Channel='time']}
EncType    := 'quantitative' | 'ordinal' | 'nominal' | 'temporal'
Condition  := [{Test<String>, Value<Any>}]
Ramp       := 'linearl' | 'exponential' | 'abrupt'   How gradually to change audio properties
Aggregate  := 'mean' | 'median' | ...
Bin        := <Boolean> | {maxbins, nice, step, exact}
By         := 'sequence' | 'overlay' | ['sequence' | 'overlay']
                  How to arrange repeated streams
TimeUnit   := 'year' | 'month' | 'day' | ...   Category (aggregate by)
TimeLevel  := 'year' | 'month' | 'day' | ...   Precision (aggregate up to)
```

7. Scale: customizing how a data variable is mapped to an auditory variable

```
Scale := {Description, Polarity, Domain, The set of the data variable to map
Different ways to specify the set of auditory values to encode data to
         (Range | MaxDistinct<Boolean> | Times<Number>
          Explicitly  Maximum audible range  By multiplying data by a factor
         | (Length<Number>)[Channel='time'])
           For a time channel, by providing the length of the stream
         (ScaleType, Zero<Boolean>)[EncType='quantitative'],
                                    Whether to include zero in the domain
         (Timing)[Channel='time'],
         (Band<Number>)[Channel='time'|'tapSpeed'|'tapCount']}
Description := Boolean | DescriptionMarkUpString
Domain      := [Any]      Whether to and how to generate the description of the channel ≈ legend
Range       := [Any]
Polarity    := 'positive'   |  'negative'
Higher data value... to higher audio value   to lower audio value
ScaleType   := 'linear' | 'log' | 'pow' | 'sqrt' | 'symlog'
                  The type of the scale function for a quantitative channel
Timing      := 'abolute' | 'relative' | 'simultaneous'
For a discrete tone, each audio point is played...   at the time specified by the mapping   one after another   all together at time 0
```

8. Tick: a sound repeating every certain time interval ≈ axis

```
Tick     := [TickItem]   This list form allows different ticks for different streams.
                Referred to by Channel.Tick
TickItem := {Name<String>, Interval<Second>, OscType,
             Pitch<Hz>, Loudness<Gain>,
             PlayAtTime0<Boolean>}   Whether to play the tick at time 0 (default = true)
OscType  := 'sine' | 'sawtooth' | 'triangle' | 'square'
Gain     := Number<[0,Infinity]>
```

9. Synth: defining a custom FM or AM synthesizer

```
Synth     := [SynthItem]   Referred to by Tone.ToneType
SynthItem := {Name<String>, SynthType,
Envelope  AttackTime<Second>, ReleaseTime<Second>,
Carrier   CarrierType<OscType>, CarrierPitch<Hz>, CarrierDetune<Detune>,
Modulator ModulatorType<OscType>, ModulatorPitch<Hz>,
          ModulatorVolume<Gain>,
Modulation (ModulationIndex<Number>)[SynthType='fm'],
          (Harmonicity<Number>)[SynthType='am']}
SynthType := 'fm' | 'am'        Detune := Number<[-1200, 1200]>
```

10. (Periodic) Wave: defining the waveform of an oscillator by using cosine and sine terms

```
Wave     := [WaveItem]   Referred to by Tone.ToneType
WaveItem := {Name<String>, Real, Imag}
Real     := [Number] Cosine terms   Imag := [Number] Sine terms
```

11. Sampling: importing external audio files as instruments

```
Sampling     := [SamplingItem]
SamplingItem := {Name<String>, Referred to by Tone.ToneType
                Sample}
Sample       := Mono<UrlString> | Octave
Sound files ...   w/o pitch like drums   w/ pitch like pinao or violin
Octave       := {C0<UrlString>, ..., C7<UrlString>}
```

12. Config: specifying configuration options

```
Config := [Key<String>: Value<Any>]
          E.g., the key "timeUnit" can have a value of "beat" or "second"
```

Notations

`A := B`	**A is defined as B.**	`(A)[B=C]`	**A is available when B is C.**	`A<[B, C]>`	**A number type A with a range between B and C.**
`{A, B}`	**A tuple of A and B.**	`A<B>`	**An item of type A.**	`[A]`	**A list of type A.**
`A | B`	**A or B.**	`*<A>`	**Anything of type A.**	`[A: B]`	**A dictionary with key of type A and value of type B.**