Koreographer

Developer Preview
Manual 0.0.7

Jeff Campbell
Eric Robinson

Contents

A Few Notes Before We Begin

There are some assumptions for the user concerning specific commands or common conventions that will be referenced throughout this document that may change depending on the operating system you are using. These will not be referenced in technical or explicit detail except where absolutely necessary.

For example, it is assumed that when asked to save a file or scene in Unity that you are either:

1. Using the Unity menu bar *File→Save* to save the scene and modified files.
2. On Windows, pressing the *CTRL+S* keys to save the scene and modified files.
3. On Mac, pressing *CMD+S* to save the scene and modified files.

Summary

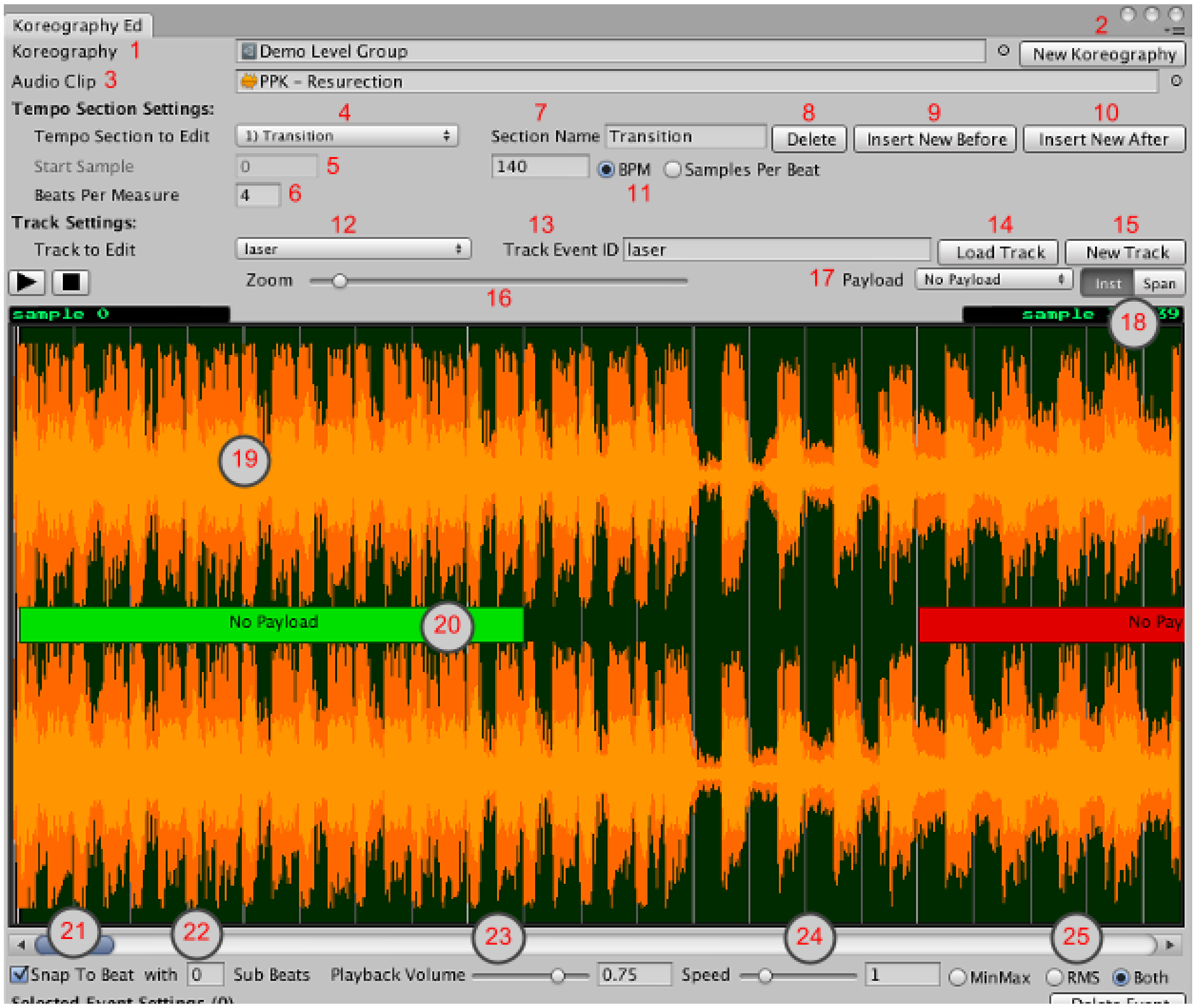
Koreographer is a Unity Editor Extension made up of a package of C# scripts that can help you fully utilize your audio to drive gameplay, visuals, etc. with an intuitive event based API.

Koreographer Overview

Koreographer is composed of two significant areas; the **Koreography Editor** and the **Koreographer** itself.

1. The **Koreography Editor** is the Unity Editor window where events can be associated with an AudioClip through the creation of *Koreography*, which itself contains zero or more *KoreographyTracks*, each specifying a series of events.
2. The **Koreographer** is the runtime engine that triggers events defined in loaded *Koreography* and, thus, *KoreographyTracks* based on the playback of the associated AudioClip. The event model used is a Subscriber-Publisher model, where a class can register as a subscriber to a specific Event ID. When an event with that ID is triggered, all subscribers are given a copy of the event, including any associated *Payload* data, as well as optional timing metadata.

Koreography Editor



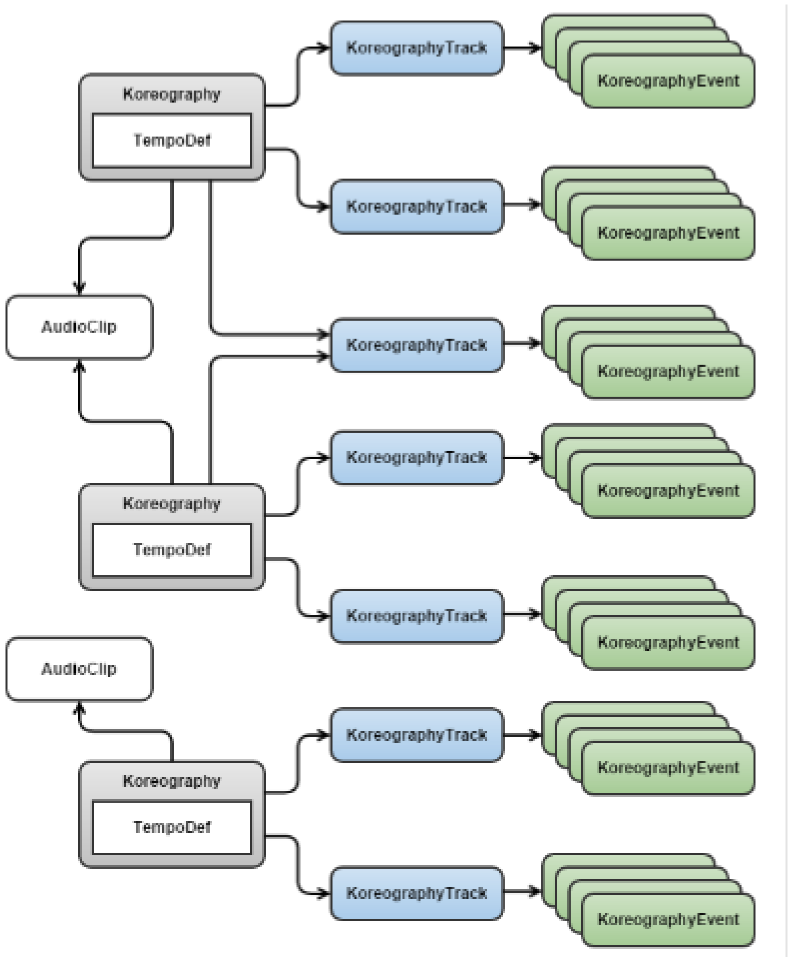
Koreography: Demo Level Group
Audio Clip: PPK - Resurection
Tempo Section to Edit: 1) Transition
Section Name: Transition
Start Sample: 0
140 [✓ BPM] Samples Per Beat
Beats Per Measure: 4
Track to Edit: laser
Track Event ID: laser
Payload: No Payload
[✓ Snap To Beat] with 0 Sub Beats
Playback Volume: 0.75
Speed: 1
MinMax RMS [✓ Both]

1. **Koreography**: The currently loaded *Koreography*.
2. **New Koreography**: Opens a dialog to create and save a new *Koreography* asset.
3. **AudioClip**: The AudioClip associated with the current *Koreography*.
4. **Tempo Section to Edit**: The currently selected Tempo Section.
5. **Start Sample**: The first sample in the current Tempo Section.
6. **Beats Per Measure**: The number of beats per measure in the current Tempo Section.
7. **Section Name**: The name of the currently selected Tempo Section.
8. **Delete**: Remove the currently selected Tempo Section.
9. **Insert New Before**: Insert a new Tempo Section before the currently selected one.
10. **Insert New After**: Insert a new Tempo Section after the currently selected one.
11. **BPM/Samples Per Beat**: The Beats Per Minute (or Samples Per Beat) of the current Tempo Section.

12. **Track to Edit**: The currently selected *KoreographyTrack* of this *Koreography*.
13. **Track Event ID**: The Event ID of the *KoreographyTrack*. You can register a subscriber for events of this ID with the *Koreographer* class.
14. **Load Track**: Load a *KoreographyTrack* into the *Koreography*.
15. **New Track**: Opens a dialog to create, save, and associate a new *KoreographyTrack* with this *Koreography*.
16. **Zoom**: Zoom controls for the audio timeline.
17. **Payload**: What payload to attach to newly created *KoreographyEvent*s.
    - **No Payload**: Newly created events will have no payload attached.
    - **Curve**: Newly created events will have a curve attached. This type is frequently used to animate properties in time with the music.
    - **Float**: Newly created events will have a Float value (number) attached.
    - **Text**: Newly created events will have a Text string attached. This can be used to embed lyrics into the music.
18. **Event Creation Mode**: Instructs the editor on how it should create new events. Either one of:
    - **Inst**: Created events are mapped (anchored) to a single sample and therefore span 0 samples. They are similar to Unity's Animation Events.
    - **Span**: Created events span 1 or more samples. They define a range along the timeline within which the event will be constantly triggered.
19. **Waveform Display**: A visual representation of the sample data at the given zoom level. The top display is the Left Channel while the bottom display is the Right Channel.
20. **KoreographyTrack Display**: Events defined in the selected *KoreographyTrack* will draw between below the Left Channel Waveform Display. In this case, two *KoreographyEvent*s are visible, both without a payload and spanning multiple samples. The one highlighted in Green is in the "Selected" state.
21. **Snap to beat**: When creating events, if checked the start/end positions of events will be set to that of the nearest beat (this is also known as quantization). If unchecked, the start/end position of events will be set to the sample position reported by the underlying AudioListener at the time input is processed (quantization is off).
22. **# of Sub-Beats (snap to beat)**: If Snap to beat is checked, the event will snap to the nearest sub beat. If this number is more than 0, it can snap to a position between major beats (as depicted by the vertical lines backing the waveform display). The number of sub beats will distribute equally between major beats.
23. **Playback Volume**: The volume of the currently selected track when playing in the *Koreography Editor*.
24. **Speed**: The speed of the currently selected track when playing in the *Koreography Editor*.
25. **Waveform Visualization:**
    - **MinMax:** Looks over the range of samples represented by the given pixel position and selects the biggest and smallest value.

- **RMS**: Looks over the range of samples represented by the given pixel position and performs a Root Mean Square calculation. The value is inverted (+ to -) to create a vertically symmetric waveform representation.
- **Both**: This overlays the RMS representation over the MinMax version. This is the standard used by Audacity and is useful as both algorithms tend to highlight different features of the underlying audio. It is also the most processor intensive.

Koreography Class Composition



Appendix

AudioClip: From Unity's Reference Manual - "Audio Clips contain the audio data used by Audio Sources. Unity supports mono, stereo and multichannel audio assets (up to eight channels). The audio file formats that Unity can import are .aif, .wav, .mp3, and .ogg. Unity can also import tracker modules in the .xm, .mod, .it, and .s3m formats. The tracker module assets behave the same way as any other audio assets in Unity although no waveform preview is available in the asset import inspector."

Beats-Per-Minute (BPM): Beats per minute or BPM is a unit typically used as a measure of tempo in music. The tempo of a piece will typically be written at the start of a piece of music, and in modern Western music is usually indicated in BPM. This means that a particular note value (for example, a quarter note or crotchet) is specified as the beat, and that the amount of time between successive beats is a specified fraction of a minute. The greater the number of beats per minute, the smaller the amount of time between successive beats, and thus faster a piece must be played.

Koreography Editor: The *Koreography Editor* is the main UI workflow tool for creating and configuring *Koreography* , *KoreographyTrack*s, and *KoreographyEvent*s.

Koreographer: The *Koreographer* is the main playback engine and is object responsible for triggering *KoreographyEvent*s. It *is* the Choreographer. It takes the current music point, queries the *KoreographyTrack*(s) for events at those times, and then sends cues to "actors" that are listening for directions.

KoreographerInterface: The *KoreographerInterface* abstraction is designed to allow users a simple integration with a pre-existing music playback solution, whether it be of their own design, an included music player, or one they got off the Asset Store. In the future, this will likely be used to create pre-built integrations for existing Asset Store packages. The two provided classes in the Koreographer package that implement this interface are the *MusicPlayer* and *SimpleMusicPlayer*.

Koreography: The *Koreography* associates a single AudioClip with one or more *KoreographyTrack*s. It is the package of data that informs the *Koreographer* about what events to generate while the associated AudioClip is playing. It also holds the Tempo Map information (Tempo Sections) that enables the *Koreographer* to provide a Music Time interface.

KoreographyTrack: A *KoreographyTrack* groups 0 or more *KoreographyEvent*s and is identified with a globally non-unique string Event ID. When registering for events with the *Koreographer*, this Event ID is used to associate the callback with events triggered from the specific *KoreographyTrack*. Currently, an Event ID *is* (or tries to be) unique within a given *Koreography* asset.

KoreographyEvent: A *KoreographyEvent* identifies a specific location in time of an AudioClip. *KoreographerEvent*s can be considered as being either:

- Instantaneous (will only trigger once per playback) or
- Span (framerate, range length, etc. dependent - potentially triggers multiple times per playback).

Payloads: Payloads are packages of data that are attached to *KoreographyEvent*s and accessed via the *KoreographEvent*.Payload property. Currently there are four available options:

- **No Payload**: No payload is associated with the event. The Payload property will return *null*.
- **Curve**: A curve (AnimationCurve) is associated with the event. The Payload property will return an object of type *CurvePayload* providing access to an AnimationCurve. When attached to an event that spans multiple samples a curve will be drawn spanning those samples. These Payloads are frequently used to animate systems/logic/assets in the scene/game.
- **Float**: A float value (number) is associated with the event. The Payload property will return an object of type *FloatPayload* providing access to the float value. When attached to an event that spans multiple samples the float value will be displayed spanning those samples.
- **Text**: Text (string) is associated with the event. The Payload property will return an object of type *TextPayload* providing access to the string value. When attached to an event that spans multiple samples, the string value will be displayed spanning those samples.

It is possible to extend the system with custom payload types.

Sample(s): In signal processing, **sampling** is the reduction of a continuous signal to a discrete signal. A common example is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal). In this context, a **sample** refers to a value or set of values at a point in time and/or space.

Put more simply, samples make up the raw audio data. Each sample is a float value between -1 and 1. These values can be thought of as positions of a speaker cone. How quickly they are fed to the speakers depends on the Sample Rate. A typical sample rate is 44100 samples per second. This means that in a single second of audio playback, the system will have processed 44100 samples. Thought of another way, each sample represents 1/44100 seconds of time, or roughly 0.0000227 seconds.

Tempo Map: The collection of Tempo Sections that define the measure and beat locations within the raw audio data.

Tempo Section: A single section of the Tempo Map that defines a start sample position within the audio data and a tempo (internally this is stored as Samples Per Beat).

Hotkeys

Key	**Usage**
A[1]	Toggles Select mode for mouse interactions with *KoreographyEvent*s
S[1]	Toggles Draw mode for mouse interactions with *KoreographyEvent*s
Z[1]	Toggles Instantaneous *KoreographyEvent* generation in the editor
C[1]	Toggles Span *KoreographyEvent* generation in the editor
⌘[2] + A	Select All Events
⌘ + X	Cut Selected Event(s) to clipboard
⌘ + C	Copy Selected Event(s) to clipboard
⌘ + V	Paste Event(s) from clipboard
⌘ + ⇧[3] + V	Paste earliest Payload from clipboard into selected Event(s)
⌘ + Z	Undo[4]
⌘ + ⇧ + Z	Redo[4]

[1] Only work when the Waveform Display has focus
[2] ⌘ = Control on Windows platforms
[3] ⇧ = Shift
[4] Does not work with *AnimationCurve*s (*Curve Payload*)