Koreographer User's Guide

for v1.6.0

Table of Contents

What is Koreographer?

Koreographer is a game development tool and Unity Plugin that simplifies the process of synchronizing music and action in your video game. It's simple editing interface allows Unity developers to map rhythms, beats, notes, volume and other dynamics of the music to events in the game. Any element of the game can be Koreographed: animations, sound effects, and even gameplay logic. Koreographer can be used to create rhythm games, make any game more cinematic, enhance game environments with music, and create new controls and music-driven gameplay.

At its core, Koreographer is an Event System. As music or sounds play in your game, the Koreographer system reports Koreography Events to any component or system registered to listen. Koreographer also uses tempo information stored in the Koreography to provide a music timeline to scripts. This music timeline can be used to power animations and other systems to keep timing feeling musical.

Installing Koreographer

Koreographer is bundled as a Unity Asset Package and is installed in the standard Asset Package fashion. There are three main ways to install Koreographer, depending on how you obtained your copy of the package.

For Asset Store purchases, see the official documentation on Importing from the Asset Store.

For other purchases, there are two equivalent installation methods: drag-and-drop or via the menus. The Drag-and-Drop method is as simple as dragging the package from your file system onto Unity's Project folder. For the Menus method, go to *Assets→Import Package→Custom Package…* and locate the Koreographer UnityPackage you acquired.

For all three methods, a package import dialog will appear. Ensure that everything is checked (the *All* button) and press the *Import* button. Unity will then install Koreographer files into the proper locations to get you moving.

Installed Components

Koreographer comes with several Components. They appear in the following order (main Koreographer components are **bold**; organizational hierarchy are *italic*):

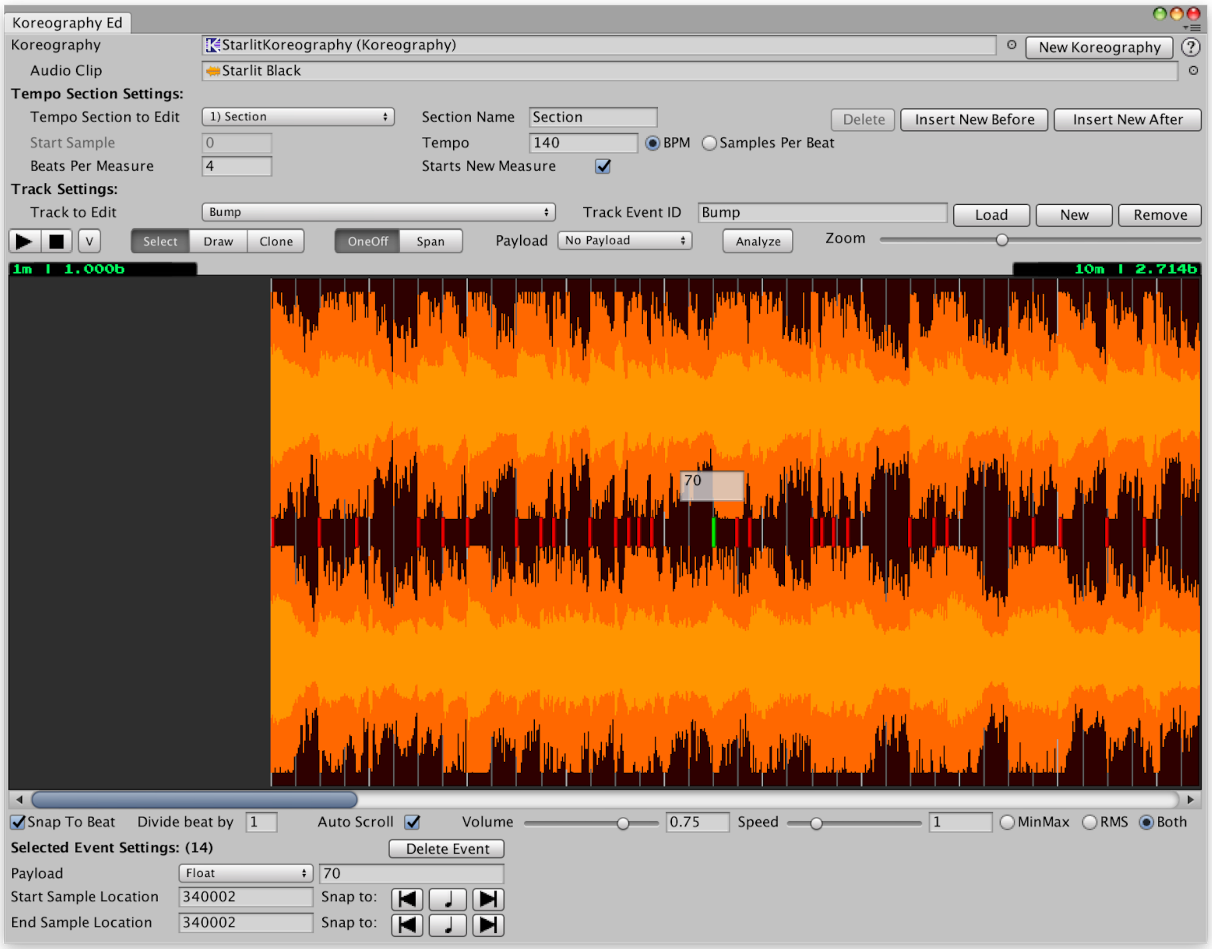
- **Koreographer**
- *Demos*
  - Pro Color Adjuster
  - Cube Scaler
  - Pro Demo Controls UI
  - Emit Particles On Span
  - Musical Impulse
  - Tempo Switch
  - UI Message Setter
- *Music Players*
  - Pro *Experimental*

        - *Sample Sync System*
            - **Audio Bus**
        - **Sample Sync Music Player**
    - **Multi Music Player**
    - **Simple Music Player**
- *Visors*
    - **Audio Source Visor**

Installed Editors

Koreography Editor

The Koreography Editor is a window that provides the interface to **view**, **edit**, and **create** your Koreography. The menu option to open the Koreography Editor is located in the main *Window* menu.



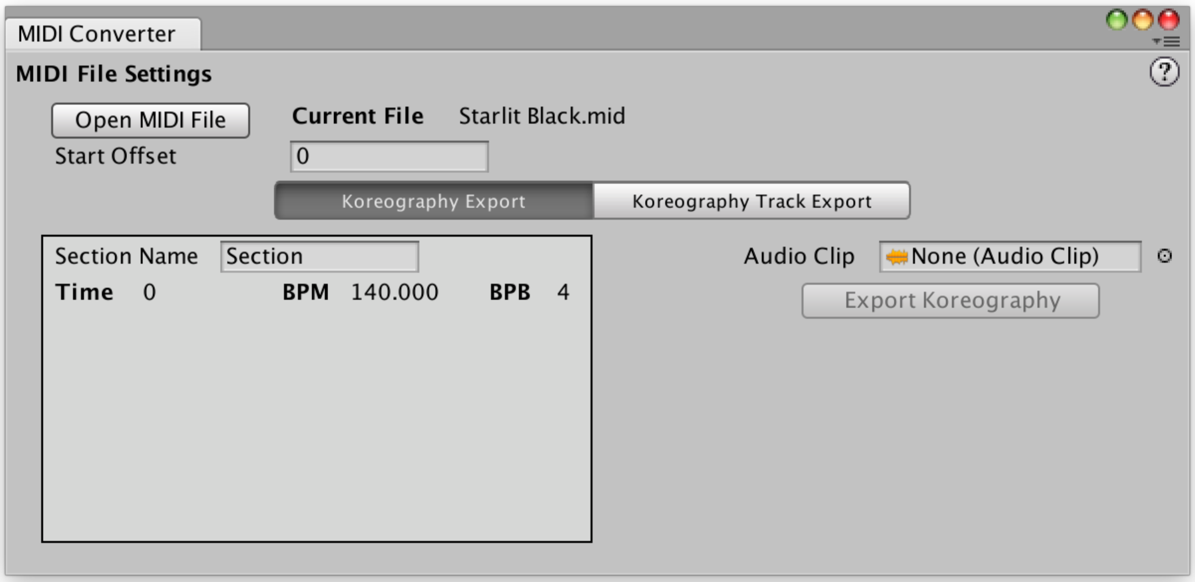
Koreography: StarlitKoreography (Koreography)
Audio Clip: Starlit Black
Tempo Section Settings:
Tempo Section to Edit: 1) Section
Section Name: Section
Start Sample: 0
Tempo: 140 [✓ BPM] Samples Per Beat
Beats Per Measure: 4
Starts New Measure: ✓
Track Settings:
Track to Edit: Bump
Track Event ID: Bump
Payload: No Payload
Snap To Beat: ✓
Divide beat by: 1
Auto Scroll: ✓
Volume: 0.75
Speed: 1
MinMax RMS [✓ Both]
Selected Event Settings: (14)
Payload: Float 70
Start Sample Location: 340002
End Sample Location: 340002

The Koreography Editor

See the Koreography Editor section for more.

Pro MIDI Converter

The MIDI[1] Converter is not a core system of Koreographer. Rather it is a system that enables an enhanced workflow for users of Digital Audio Workstations, or people with access to an accompanying MIDI file. For Koreographer Pro users, the menu option to open the MIDI Converter is located in the main *Window* menu.


MIDI Converter

MIDI File Settings

Open MIDI File   Current File   Starlit Black.mid

Start Offset   0

Koreography Export | Koreography Track Export

Section Name   Section

Time 0   BPM 140.000   BPB 4

Audio Clip   None (Audio Clip)

Export Koreography

The MIDI Converter

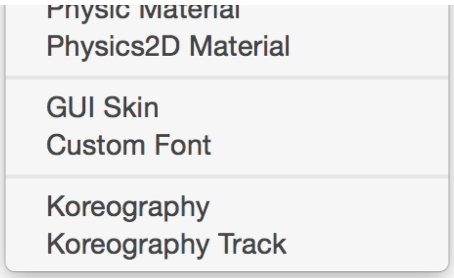
See the **MIDI Converter Overview** document included with this documentation for more.

Asset Menu Items

The Koreographer extension adds two Assets to Unity:

1. Koreography
2. Koreography Track

These can be found in the Create Assets menu located at *Assets→Create*.



Koreography and Koreography Track assets in the “Create Asset” menu

[1] Musical Instrument Digital Interface - MIDI files are a standardized way for musical sequences to be saved, transported, and opened in other systems.

Demo Content

Koreographer ships with multiple demos. One example (the **Basic Demo**) is installed by default, while several others (**Feature and Mechanic Demos**) must be installed from Asset Packages. See below for more details on each.

> **Note:** All demo content scripts exist in the `SonicBloom.Koreo.Demo` namespace.

Basic Demo

This demo requires no extra steps to install and can be found directly in the *`Koreographer/Demos`* folder. This demo contains a handful of simple scripts, scenes, and support assets that show off the basics of working with Koreographer. The scenes were designed to be viewed in succession, with each expanding on the contents of the previous in the series. The scripts are well-commented and have been kept intentionally simple to provide easy-to-understand example use cases.

Feature and Mechanic Demos

In addition to the Basic Demo, Koreographer contains several other demos that provide a working implementation of a specific mechanic or use. These demos come fully documented with both inline script comments and a demo-specific manual (manuals will be installed in the *`Koreographer/Documentation`* folder). These demos are packaged as Asset Packages and can be found in the *`Koreographer/Demos`* folder.

Koreographer contains the following feature and mechanic demos:

- **Karaoke Demo:** Contains a working karaoke text-painting example.
- **Rhythm Game Demo:** Contains a basic example of the classic rhythm gameplay mechanic.
- Pro **Custom Payload Demo:** Contains example implementations of custom payload types and their support Koreography Track types. Specifically:
    - **Material Payload example:** showcases a simple custom payload implementation.
    - **MIDI Payload example:** showcases a custom payload implementation and a Koreography Track implementation which hooks into the MIDI Converter's custom MIDI conversion functionality.

> **Note:** Due to API compatibility issues, these demos are only available in Koreographer for Unity 5.0.0 and newer.

Integration Packages

Koreographer comes with support for several popular third party tools. Integration packages are delivered as Asset Packages and can mostly be found in the *`Koreographer/Integrations`* folder. Each integration package contains its own manual (manuals will be installed in the *`Koreographer/Documentation`* folder). Due to compatibility issues with third party package versions, not all integrations are available for all versions of Unity supported by Koreographer.

Koreographer contains the following integration packages:

- **PlayMaker Integration:** Provides an integration with the popular PlayMaker visual scripting extension by Hutong Games.
- Pro **Fabric Integration:** Provides an integration with Tazman Audio's Fabric audio system.

    - Requires Koreographer for Unity 4.6 or newer.
- Pro **SECTR Audio Integration:** Provides an integration with Make Code Now!’s SECTR Audio extension.
- Pro **Master Audio Integration:** Provides an integration with Dark Tonic’s Master Audio extension.

In addition, the following integration packages are available upon request:

- Pro **Wwise Integration:** Provides an integration with Audiokinetic’s Wwise audio system in Unity.

If you would like an integration with a popular third party tool not in available in this list, please feel free to reach out to us with a feature request.

Koreographer System Breakdown

Koreographer consists of three major pieces: the **Data**, the **Editor UI**, and the **Runtime**. Koreography Data stores the information about musical event timing that you wish your systems to be informed about. You create and edit that information using the Editor UI and related tools. Finally, the Runtime system loads the information you prepared and uses it to generate streams of events for which other systems in your game can register.

This section will go into each of these pieces in more depth.

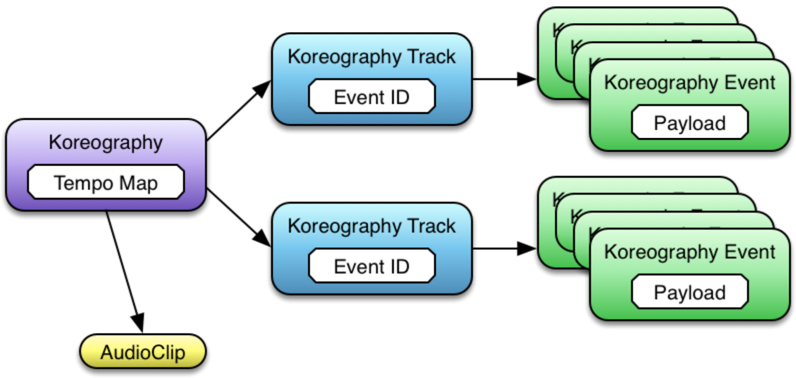
Koreography Data

Data, henceforth called *Koreography*, is at the core of Koreographer. It is a map that describes information about a piece of audio - information that your game or application can use to enhance the experience you’re creating.

This section describes the contents of Koreography; it is the legend for the Koreography map.

Koreography

In Koreographer, the collection of information or directions that you create about audio is called Koreography. Koreography consists of a reference to a piece of music (in Unity, this is an AudioClip reference by default), a Tempo Map, and a set of Koreography Tracks, which contain an Event ID and a set of Koreography Events, which in turn may optionally contain a Payload. The following diagram illustrates how all of these pieces fit together:



Koreography is serialized as a Unity Asset file. The only data stored in this file is the Tempo Map, the AudioClip reference, and references to any included Koreography Tracks, which are also serialized as separate Unity Asset files.

Each piece is discussed in further detail below.

AudioClip

The AudioClip reference is what connects the Koreography to a specific audio file. The Koreographer system gets some metadata required for runtime processing of the Koreography data from the AudioClip. In the Editor, the AudioClip provides the sample data required to draw the waveform visualization. At runtime, the Koreographer system uses the name of the AudioClip to determine whether the target audio is playing and, if so, the timing necessary to trigger events.

Load Type in the Editor

Unity can load AudioClips in your game in various ways. This can interfere with Koreographer's operation ***in the Editor***. A major requirement for drawing the waveform is access to the underlying audio data. Unity allows access to this data when the Load Type is set to Decompress On Load. By default, however, imported AudioClips have a Load Type of Compressed In Memory. The Koreography Editor will detect this setting, display a warning in lieu of the waveform, and then offer to make the setting change for you.

The Load Type setting discussed in the previous paragraph is *only* of concern for the Koreography Editor. The Load Type has no effect on Koreographer's ability to track audio position at runtime.

Tempo Map

While Koreography can be used to map out the contents of any audio, there are special affordances built into the system for music. Specifically, the Tempo Map allows you to add information about the rhythm of a piece of music.

The Tempo Map in Koreography is a list of Tempo Sections. Each section defines the "start point for" and the "beats-per-minute (BPM) of" a section of the audio. Most music should only require a single entry in the Tempo Map.

Koreography Tracks

A Koreography Track is a sequence of Koreography Events that is paired with an Event ID, both described below. Koreography Tracks are the source for the streams of events that Koreographer provides your game at runtime. They typically refer to a single feature or *feeling* of a piece of audio. You could use a Koreography Track to describe the "lyrics" of a song or speech, the "bass beat" of a piece of music, or some "overdrive" feeling that you want to track with a special effect or gameplay system (music-driven overdrive anyone?). Note that "lyrics", "bass beat", and "overdrive" would all be great Event ID candidates.

Koreography Tracks are serialized as their own Unity Asset files, separate from the Koreography Asset files. The data stored in a Koreography Track Asset file includes the Event ID and the Koreography Events with any Payloads they may have.

Event ID

The Event ID is a human readable name that you give to the collection of Koreography Events defined within a Koreography Track. This value is set in the editor and can be different from the name of the Koreography Track asset file.

At runtime the Event ID identifies a stream of Koreography Events for which any registered system could listen. For example, a UI system might register for “Lyric” events. This system would then receive notifications from any triggering Koreography Track with the “Lyric” Event ID.

> **Note:** Currently, Event IDs must be unique within Koreography (you cannot load more than one Koreography Track with any given Event ID into a single Koreography).

Koreography Events

Koreography Events are the flexible core at the heart of the Koreographer system. You add these to a Koreography Track in the editor and register to listen for them at runtime. They contain the timing and extra metadata about a specific moment of audio. For example, you could create a Koreography Event at the exact moment of a beat drop and a Payload with the number “11” in it because that’s how big the beat drop was!

Koreography Events come in two related varieties: the OneOff and the Span. These are described below.

Span Events

A Span has distinct start and end times within the audio timeline. These times define a “span” of audio time. At runtime, Span Events trigger on every frame in which the audio system played any portion of the “span” of audio time.

It is possible to create a Span Event that lasts the entire length of an audio file. Such Span Events trigger every frame during the playback of the associated audio file.

OneOff Events

A OneOff is defined solely by a start time. OneOff events trigger when the playback of the associated audio file reaches the OneOff event’s location.

> **Note:** Currently, no two OneOff Events within a single Koreography Track can share the same start time.

> **Note:** Internally, the OneOff *also* has an End Time - it is simply set to the same value as the Start Time.

Payloads

Koreography Events may ***optionally*** contain a Payload. Payloads allow you to associate specific metadata to a Koreography Event. This flexibility allows you to provide extra information to the systems in your game beyond a simple notification that something is happening. Built-in Payload types include:

- Pro **Asset:** A Unity Asset.
- Pro **Color:** A Unity Color.
- **Curve:** A Unity AnimationCurve.

- **Float:** A floating point number.
- Pro **Gradient:** A Unity color Gradient.
- **Int:** An integral (whole) number.
- Pro **Spectrum:** A set (or sets) of numbers approximating an evaluated Frequency Spectrum.
- **Text:** A custom text string.

While the **Asset, Color**, **Gradient**, and **Spectrum** Payload types can only be created in Koreographer Pro versions of the Koreography Editor, Koreography data containing these Payloads can be used at runtime with any version of Koreographer. The **Spectrum** Payload is created with the Analysis functions.

The **Curve**, **Gradient**, and **Spectrum** payloads have built-in accessors for retrieving their value at a given time. When attached to a Span Event, these special accessors can help you animate float values, a color values, and more in time with the associated audio.

Payloads are accessed at runtime by your registered game systems using the Koreography Event script API.

Pro *Custom Payloads*

If the built-in Payload types do not suit your needs, you may create your own custom types. The process for creating your own custom payloads is outlined in the Custom Payload Demo, which includes a working example: the **Material** Payload type.

Creating Koreography

Koreography and Koreography Track asset files can be created with either the Koreography Editor or the included Asset Menu options. The Koreography Editor is the main interface for setting the Tempo Map, the AudioClip, creating Koreography Events, and setting Payloads.

Advanced Topics

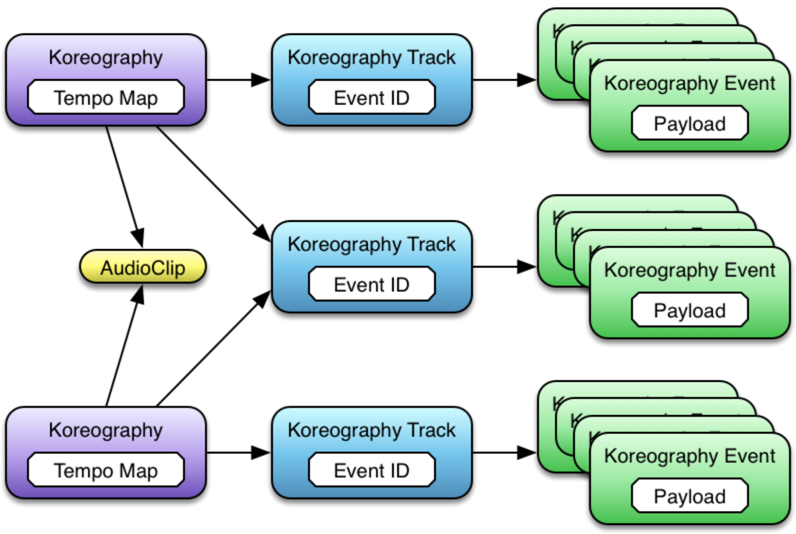
This section outlines more advanced uses and structures for Koreography data.

Shared Koreography Tracks

Koreography Tracks are stored as references within Koreography, not copied. This allows them to be shared between Koreography. This flexibility can be leveraged in a variety of ways. For example:

- **Difficulty-based Koreography** - You can create different Koreography that have varying combinations of Koreography Tracks. One set of shared Koreography Tracks may be used to drive background visuals, while another set of difficulty-specific tracks could drive gameplay.
- **Remixes with Instrumental Variation** - You can create different Koreography for audio remixes. If you have three variations of a piece of music all with the same bass line, you could create one Koreography for each AudioClip, and use the same bass line-matched Koreography Track in each of them.

The following diagram illustrates how the data may be connected as in the **Difficulty-based Koreography** example:



Two Koreography packs sharing a Koreography Track

In the diagram above, the two Koreography packs reference the same AudioClip. There is no requirement for this.

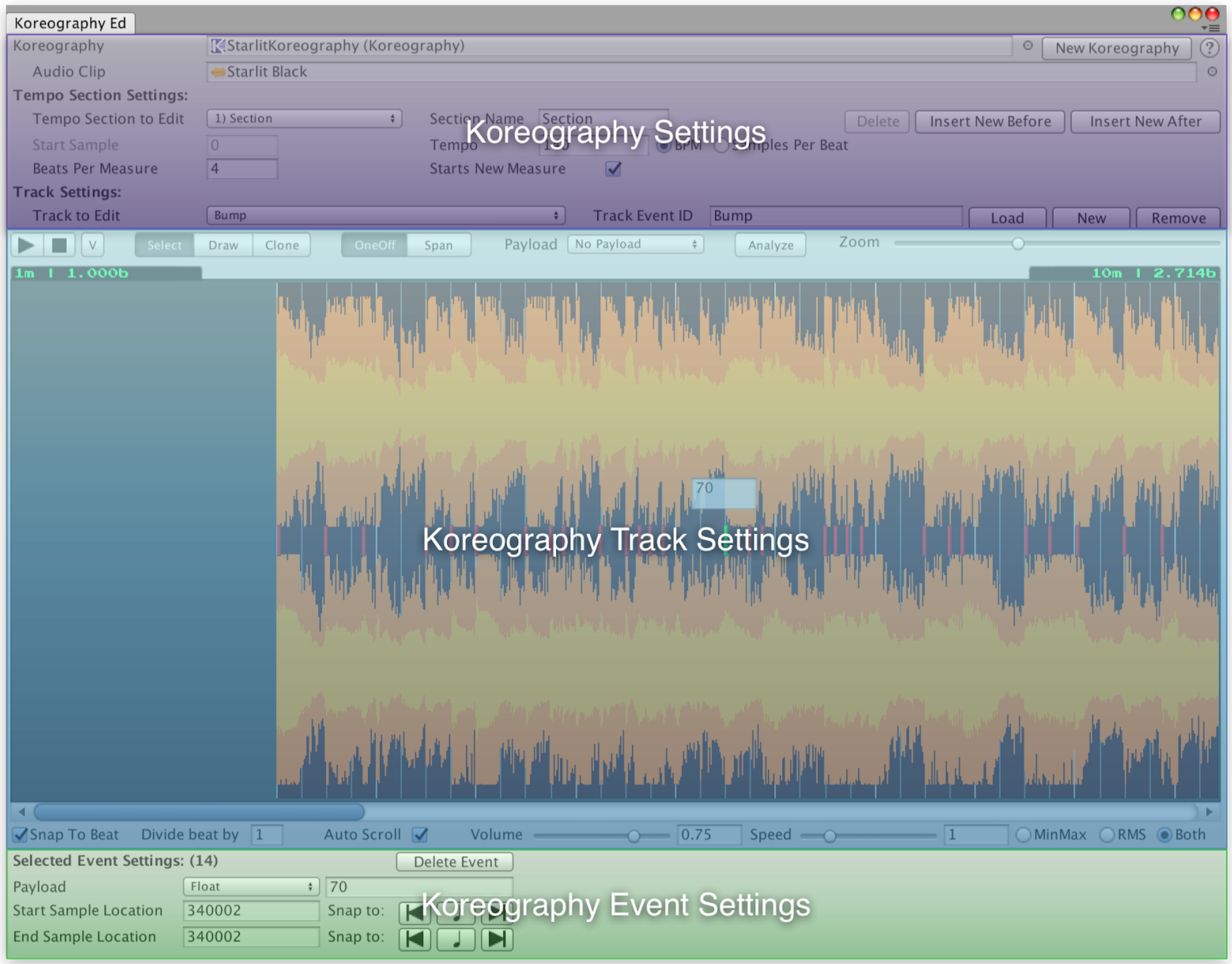
This list is by no means comprehensive: how can you make use of this flexibility?

Koreography Editor

The Koreography Editor is the tool that enables you to configure Koreography Data. This section will cover aspects of the Editor Window and its built-in functionality.

User Interface Overview

The main Koreography Editor is broken into three main sections. These sections allow you to control Koreography settings, Koreography Track settings, and Koreography Event settings.



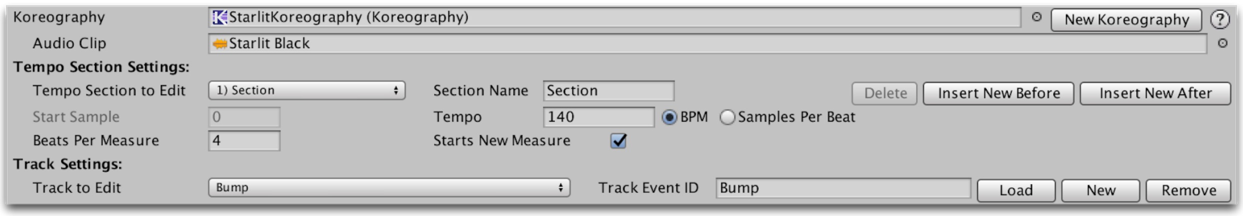
Koreography: StarlitKoreography (Koreography)
Audio Clip: Starlit Black
Tempo Section Settings:
Tempo Section to Edit: 1) Section
Section Name: Section
Start Sample: 0
Beats Per Measure: 4
Starts New Measure: ✓
Track Settings:
Track to Edit: Bump
Track Event ID: Bump
Payload: No Payload
1m | 1.000b
10m | 2.714b
Snap To Beat: ✓
Divide beat by: 1
Auto Scroll: ✓
Volume: 0.75
Speed: 1
MinMax RMS [✓ Both]
Selected Event Settings: (14)
Payload: Float 70
Start Sample Location: 340002
End Sample Location: 340002

Overview of Main Koreography Editor sections

These sections are outlined in greater detail in the following sections.

Koreography Settings

Fields and controls in this section allow you to create, edit, and manage Koreography settings. The UI is not fully enabled until a Koreography is loaded or created.



Koreography: StarlitKoreography (Koreography)
Audio Clip: Starlit Black
Tempo Section Settings:
Tempo Section to Edit: 1) Section
Section Name: Section
Start Sample: 0
Tempo: 140 [✓ BPM] Samples Per Beat
Beats Per Measure: 4
Starts New Measure: ✓
Track Settings:
Track to Edit: Bump
Track Event ID: Bump

Koreography Settings

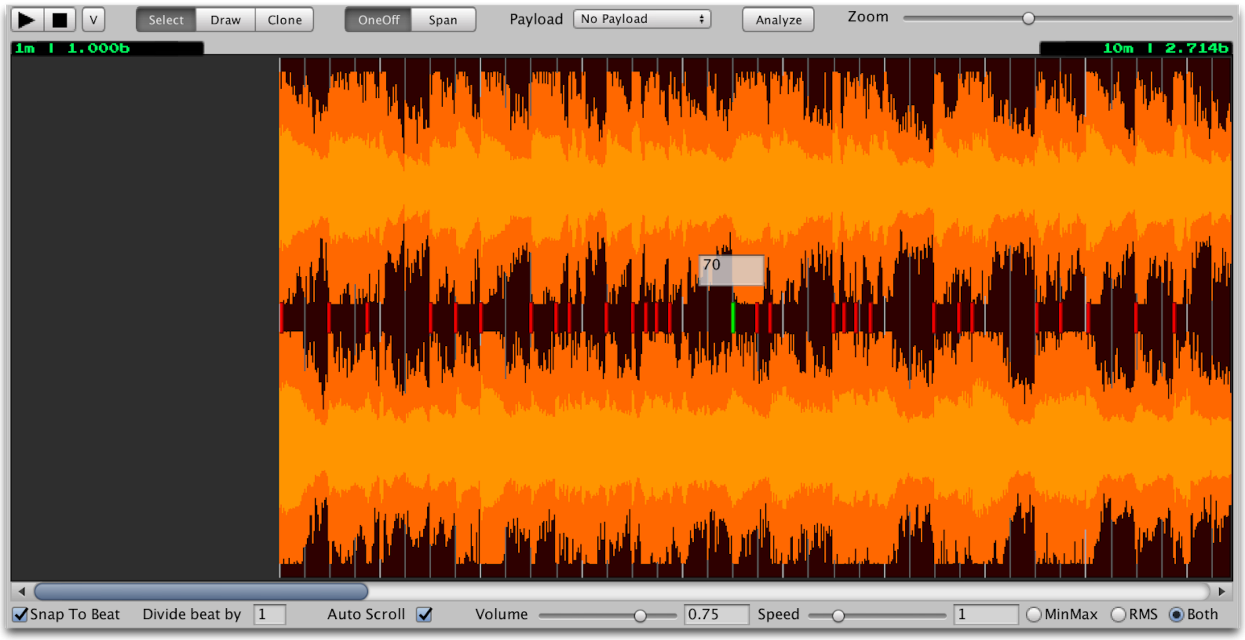
Individual elements are discussed below. These are discussed in top-to-bottom, left-to-right order.

1. **Koreography:** The currently loaded Koreography.
2. **New Koreography:** Opens a dialog to create and load a new Koreography asset.
3. **“?”:** Opens the Koreographer Help panel.

4. **AudioClip:** The AudioClip associated with the currently loaded Koreography.
5. **Tempo Section Settings:** Settings that define/configure the Tempo Map associated with the Koreography's audio.
    a. **Tempo Section to Edit:** The currently selected Tempo Section.
    b. **Section Name:** The name of the currently selected Tempo Section. You may rename the currently selected section with this control. Tempo Section Names do not have to be unique.
    c. **Delete:** Remove the currently selected Tempo Section. Tempo Sections may only be removed if more than one exists.
    d. **Insert New Before:** Insert a new Tempo Section before the currently selected one.
    e. **Insert New After:** Insert a new Tempo Section after the currently selected one.
    f. **Start Sample:** The first sample of audio data in the current Tempo Section. The first Tempo Section must have a Start Sample of 0.
    g. **Tempo:** The musical speed of the Tempo Section. Typically this is calculated as Beats Per Minute.
    h. **BPM/Samples Per Beat:** Determines how the tempo value should be shown - either Beats Per Minute or Samples Per Beat.
    i. **Beats Per Measure:** The number of Beats Per Measure (or Beats Per Bar) in the current Tempo Section.
    j. **Starts New Measure:** Whether or not this Tempo Section begins a new measure (bar). This will effectively reset the beat count immediately. This is useful for abrupt transitions, cuts, or when accounting for silence at the beginning of a file.
6. **Track Settings:** Settings and controls that create/configure Koreography Tracks, particularly in relation to Koreography.
    a. **Track to Edit:** This dropdown will list all Koreography Tracks that are loaded into the current Koreography. Koreography Tracks here are identified by their Event ID, not the name of the asset file. The contents of the selected Koreography Track are shown and editable within the Koreography Track Settings section of the UI.
    b. **Track Event ID:** The Event ID of the currently selected Koreography Track.
    c. **Load:** Load an existing Koreography Track into the Koreography.
    d. **New:** Opens a dialog to create a new Koreography Track and associate it with the current Koreography.
    e. **Remove:** Remove the currently selected Koreography Track from the current Koreography. This does **not** delete the asset file.

Koreography Track Settings

Fields, controls, and displays in this section enable you to create, edit, and visualize the contents of the Koreography Track selected in the Track to Edit field. This section also contains controls that allow you to control the playback of the AudioClip to assist in editing. See the Keyboard Shortcuts section for some extremely handy controls.



Koreography Track settings and AudioClip Playback/Visualization controls

Waveform View

Visually, this section is dominated by the view of the audio waveform. The Koreography Events themselves appear between the two channels of audio: the *Left channel* appears above; the *Right channel* below. Mono AudioClips will show their single channel in place of the *Left channel*, above the Koreography Events. AudioClips with more than two channels will only show the Left and Right channels (other channels are currently ignored).

Koreography Event display differs slightly between OneOff and Span events. OneOff events are *always* depicted as little red bars (as in the screenshot above). OneOff Payloads are visible in this area when hovered over by a mouse or selected (as in the screenshot above). Span events are depicted as rectangles (not shown in the screenshot above). Their Payload, if any, is visible as the contents of the rectangle. Span Payloads can be edited by double-clicking the Span event. Payloads for any configuration of Koreography Event can be viewed and edited in the Koreography Event Settings that appear when an event is selected.

Selected events are always highlighted with a green tint.

Playback Anchor

The Playback Anchor determines the position to which the audio will reset when audio playback is stopped (via the Stop button or by reaching the end of the file). This feature allows quick auditioning of a specific location in the audio, especially when paired with Keyboard Shortcuts for playback control. The Playback Anchor can be set and cleared using the Context Menu.

Audio Scrubbing

The waveform can be “scrubbed”, allowing you to hear the audio at the position of the mouse as you drag it across the waveform. To enable this feature, use the Keyboard Shortcut for Audio Scrubbing appropriate for your operating system.

Transport Displays

Three Transport Displays provide information about the status of the currently visible waveform. The two green readouts on the left and right show the time of the left-most and right-most edge of the display, respectively. Together they indicate the bounds of the currently visible region of the waveform. The blue readout may not be visible if the audio is stopped (as in the screenshot above). It appears in the center and contains the current position of the audio playhead.

The Transport Displays are capable of displaying timing information in three formats:

1. **Music Time:** [Default] Time is shown in measures and beats, based on the Tempo settings.
2. **Solar Time:** Time is shown in hours, minutes, and seconds.
3. **Sample Time:** Time is indicated by the exact sample position of the display edges and playhead position.

Clicking on a Transport Display will cycle between these modes.

Creating Events

There are two main ways to create Koreography Events in Koreographer: drawing them with the mouse or adding them in realtime during playback using specific key commands. If a Tempo Map is configured, the Snap To Beat and Divide beat by features can be used to aid in rapid creation of well-timed events!

UI Elements

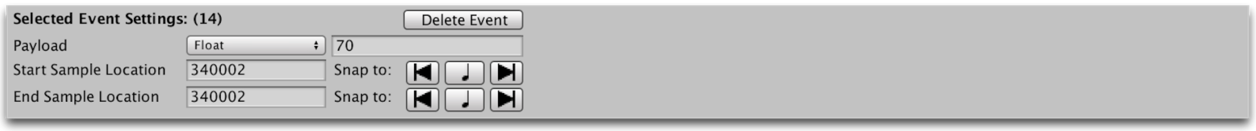
Individual UI elements are discussed below in top-to-bottom, left-to-right order.

1. **Playback Controls:** These two buttons control the basic playback state of the AudioClip.
    a. **Play (▶)/Pause (❚❚):** This button has two modes, depending upon the current state of the AudioClip. The button will read Pause if the audio is playing, Play otherwise.
    b. **Stop (■):** This button will stop AudioClip playback in any state, returning the playhead to either the start position or, if active, the Playback Anchor position.
2. **V:** Open the Event Visualizer window.
3. **Event Interaction Mode:** Determines how mouse input on the waveform display should be interpreted. All modes respect the Snap To Beat settings. Either one of:
    a. **Select:** The mouse will add or subtract events from a selection. Double-clicking the mouse on the waveform will create a OneOff event at the current mouse location.
    b. **Draw:** The mouse will add an event or events at the mouse location. This respects the Event Creation Mode.
    c. **Clone:** If no selection exists, the mouse will work as it does in the Select mode. If a selection exists, then the selected events will be duplicated (cloned) at the position of the mouse click.
4. **Event Creation Mode:** Instructs the editor on how it should create new events. Either one of:
    a. **OneOff:** Created events are mapped (anchored) to a single sample and therefore span 0 samples. They are similar to Unity's Animation Events.
    b. **Span:** Created events span 1 or more samples. They define a range along the timeline within which the event will be constantly triggered.
5. **Payload Mode:** What payload to attach to newly created Koreography Events. Options include No Payload, Asset, Color, Curve, Float, Gradient, Int, and Text.
6. Pro **Analyze:** Opens the audio analysis window.

7. **Zoom:** Zoom controls for the audio timeline.
8. **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, no quantization occurs and the start/end position of events will be set to the sample position reported by the underlying AudioSource (for event creation during playback) or at the precise position indicated by the mouse (for Draw mode). For most operations, this setting may be temporarily inverted by holding the SHIFT (⇧) key.
9. **Divide beat by:** This setting will create extra subdivisions of the beat, which is handy if you want to set an event directly between two beats. If Snap To Beat is checked, the event will snap to the nearest subdivision. If this number is greater than 1, subdivision lines will be drawn to the grid representing new snap positions.
10. **Auto Scroll:** Instructs the Koreography Editor whether to scroll the waveform display in time with the audio playback, or leave it at its current position.
11. **Playback Volume:** The volume of the AudioClip during playback.
12. **Speed:** The speed of the AudioClip during playback. This is implemented as pitch.
13. **Waveform Visualization Controls:** These settings control how the waveform should be visualized. Options include:
    a. **MinMax:** Looks over the range of sample represented by the given pixel position and selects the biggest and smallest value to display.
    b. **RMS:** Looks over the range of samples represented by the given pixel position and performs a Root Mean Square calculation. The resulting value is inverted (+ to -) to create a vertically symmetric waveform representation.
    c. **Both:** [Default] 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 has the greatest impact on performance.

Koreography Event Settings

This section is blank unless one or more Koreography Events are selected. Configurable details about the event(s) will appear in this section. When a single event is selected, a number appears in parenthesis to the right of the "**Selected Event Settings**" heading. This number is the index position of the selected event within the Koreography Track's internal list.



Selected Event Settings: (14)

Payload	Float	70	
Start Sample Location	340002	Snap to:	
End Sample Location	340002	Snap to:	

Koreography Event settings for a single selected Koreography Event

Note: The UI changes slightly when multiple events are selected. In this case, only a single Start Sample Location is shown. This represents the position of the earliest selected event. Adjusting this value will move all events equally.

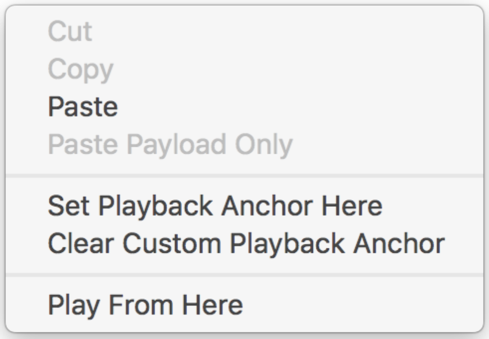
Individual UI elements are discussed below. These are discussed in top-to-bottom, left-to-right order.

- **Delete Event:** Deletes the selected Koreography Event(s).
- **Payload:** The optional payload of selected Koreography Event(s). If any payload type is selected, its value can be edited in the field that appears to the right of this control.

- **Start Sample Location:** The sample location that indicates when the Koreography Event begins along the audio timeline.
    - **Snap To:** These buttons will snap (quantize) the start sample location to the nearest, button-specific location.
        - **⏮:** Snaps to the **previous** beat/subdivision.
        - **♩:** Snaps to the **nearest** beat/subdivision.
        - **⏭:** Snaps to the **next** beat/subdivision.
- **End Sample Location:** The sample location that indicates when the Koreography Event ends along the audio timeline. This control (and related) are not available when more than a single event is selected.
    - **Snap To:** These buttons will snap (quantize) the end sample location to the nearest, button-specific location.
        - **⏮:** Snaps to the **previous** beat/subdivision.
        - **♩:** Snaps to the **nearest** beat/subdivision.
        - **⏭:** Snaps to the **next** beat/subdivision.

Context Menu

A right-click on the Waveform View or a Koreography Event will open the Koreography Editor Context Menu.



Koreography Editor Context (right-click) Menu

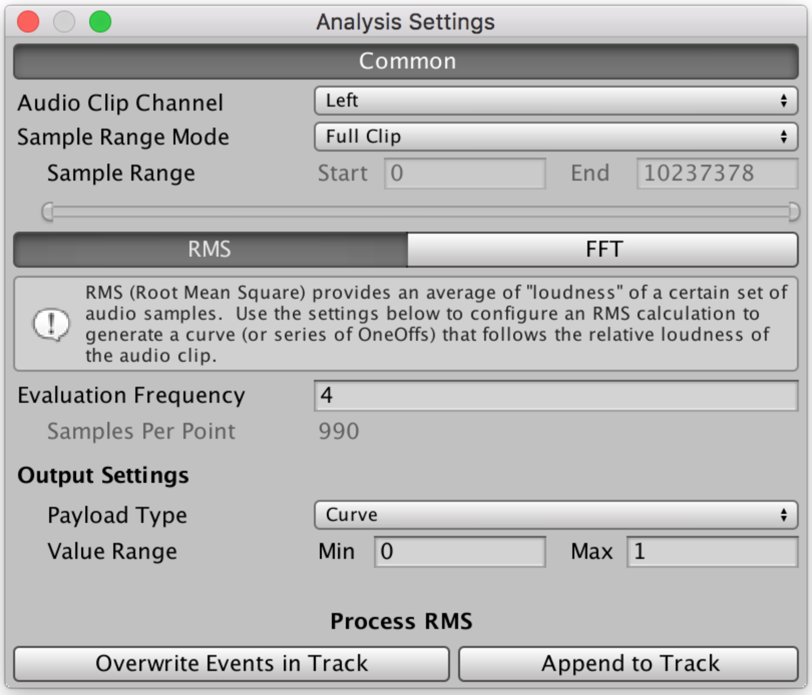
The options shown are divided into three sections: Koreography Event Editing and Audio Playback.

- Koreography Event Editing
    - **Cut:** Removes the selected Koreography Event(s) and copies them to the Koreography Editor's internal clipboard.
    - **Copy:** Copies the selected Koreography Event(s) to the Koreography Editor's internal clipboard.
    - **Paste:** If one or more Koreography Events are selected, overwrite the selection with the contents of the Koreography Editor's internal clipboard. The start location is set to that of the earliest Koreography Event in the selection. If no Koreography Events are selected, paste a fresh copy of the contents of the clipboard at the current mouse location.
    - **Paste Payload Only:** Overwrite the payload of all selected Koreography Events with a copy of that of the *earliest* Koreography Event copied to the Koreography Editor's internal clipboard.
- Playback Anchor
    - **Set Playback Anchor Here:** Sets the Koreography Editor's Playback Anchor to this location.

    - **Clear Custom Playback Anchor:** Clears the custom Playback Anchor location, resetting it to the beginning of the AudioClip. This option is only available when a Playback Anchor is set.
- Audio Playback
    - **Play From Here:** Jump the playhead to the right-click location and begin audio playback from this location.

Pro Audio Analysis

Koreographer Pro provides some basic Audio Analysis tools that assist in the creation of Koreography Events and Payloads. The audio Analysis Settings window is opened by pressing the Analyze button above the Waveform View.



Analysis Settings

Common

Audio Clip Channel: Left
Sample Range Mode: Full Clip
Sample Range: Start 0 End 10237378

RMS | FFT

RMS (Root Mean Square) provides an average of "loudness" of a certain set of audio samples. Use the settings below to configure an RMS calculation to generate a curve (or series of OneOffs) that follows the relative loudness of the audio clip.

Evaluation Frequency: 4
Samples Per Point: 990

Output Settings
Payload Type: Curve
Value Range: Min 0 Max 1

Process RMS
Overwrite Events in Track | Append to Track

The Analysis Settings window

Note: Both a Koreography and a Koreography Track must be loaded into the Koreography Editor in order for the Analysis Settings window to show as above. If either of those are not set, a warning will be displayed instead.

Common Configurable Fields

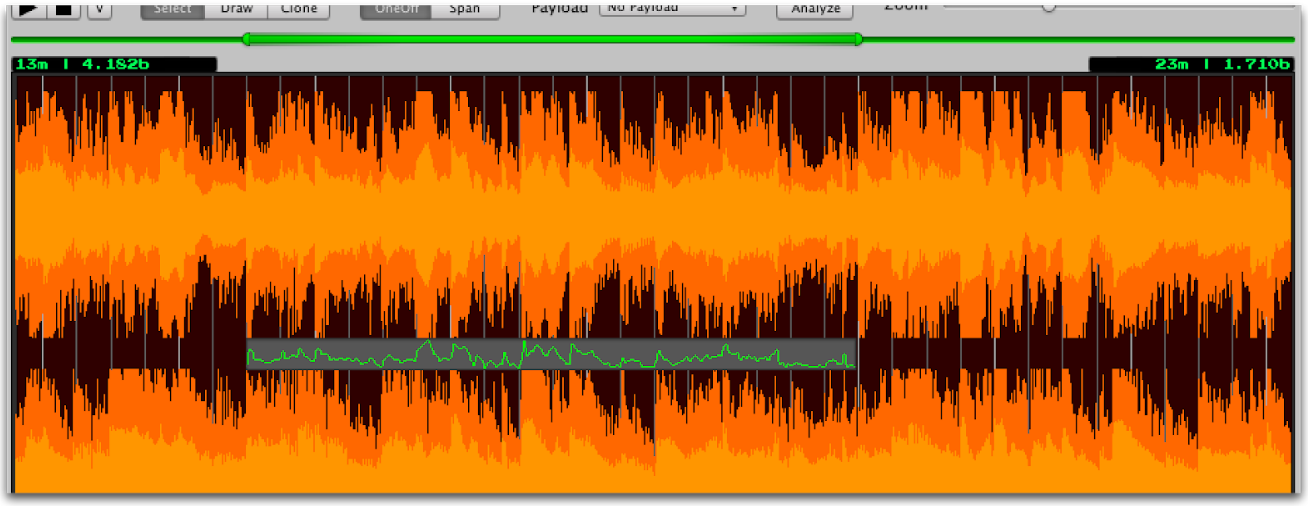
The following fields are common across all analysis methods.

- **Audio Clip Channel:** Which channel of the Audio Clip should be sourced for analysis calculations.
- **Sample Range Mode:** Determines what portion of the Audio Clip's timeline should be targeted for processing. Provides the following modes:
    - *Full Clip* - Uses the entire Audio Clip.
    - *Editor Range* - Adds a range specifier to the Koreography Editor and uses it. See Using the Editor Range UI for more.
    - *Custom Range* - Uses the **Sample Range** controls to set a sample range for configuration.

- **Sample Range:** Sets the start and end samples of the range of Audio data to process when **Sample Range Mode** is set to *Custom Range*. Otherwise, shows the current sample range.

Using the Editor Range UI

It is occasionally advantageous to specify the Sample Range visually in the Koreography Editor, making use of the Waveform visualization itself. To that end, the Analysis Settings window allows you to specify the Editor Range *Sample Range Mode* option. This will add a green min-max slider to the Koreography Editor above the Waveform view, as in the screenshot below:



Detail of the Editor Range Selector UI with Generated Output Results

By adjusting the two ends of the slider you can finely tune the Waveform area you wish to analyze. This UI will not change as the Waveform is scrolled. Scrolling the Waveform will rather change the selected sample range specified by the slider.

Note: Presently the zoom level will affect the results of the RMS analysis. If the range you wish to analyze at a given zoom level does not fit in the waveform view, either resize the Koreography Editor window or specify a specific sample range using the Custom Range Sample Range Mode.

Generating Events with Analysis Settings

The two buttons at the bottom of the Analysis Settings window allow you to specify how Koreography Events are generated using the configured settings. It should be noted that the target Koreography Track for generated Koreography Events is the Koreography Track being actively edited in the Koreography Editor.

Use the following two buttons to determine how Koreography Events are added to the Koreography Editor's currently selected Koreography Track:

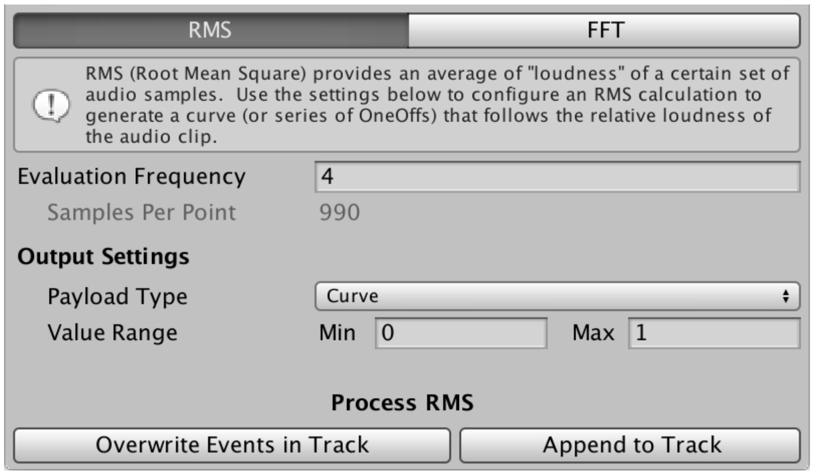
- **Overwrite Events in Track:** Overwrite all Koreography Events in the currently selected Koreography Track in the Koreography Editor with generated output.
- **Append to Track:** Append the generated analysis output to the currently selected Koreography Track in the Koreography Editor.

Koreography Event creation conducted through the Analysis Settings window properly work with Unity's Undo system. If generated Koreography Events do not match with your expectations, an efficient workflow for getting desired results involves undoing the creation of those events, adjusting the settings, and trying the output again.

RMS Analysis

This audio analysis method uses RMS calculations to create Koreography Events with configurable Payloads. As RMS is good at finding a relative loudness of an audio stream over time, this data is particularly useful for creating a speaker effect when visualized. Because this is an average of the volume across all sounds in the source audio clip, results are most effective when applied to data containing very few “voices”, such as speech or stem files.

RMS is a well-known processing algorithm and is used to generate the RMS layer in the Waveform View. By setting the **Evaluation Frequency** parameter to 1 and outputting a Payload Type of Curve, the generated RMS data will match the [top half of the] visualization in the selected channel.



RMS (Root Mean Square) provides an average of "loudness" of a certain set of audio samples. Use the settings below to configure an RMS calculation to generate a curve (or series of OneOffs) that follows the relative loudness of the audio clip.

Evaluation Frequency: 4
Samples Per Point: 990
Output Settings
Payload Type: Curve
Value Range: Min 0 Max 1

RMS Analysis Settings

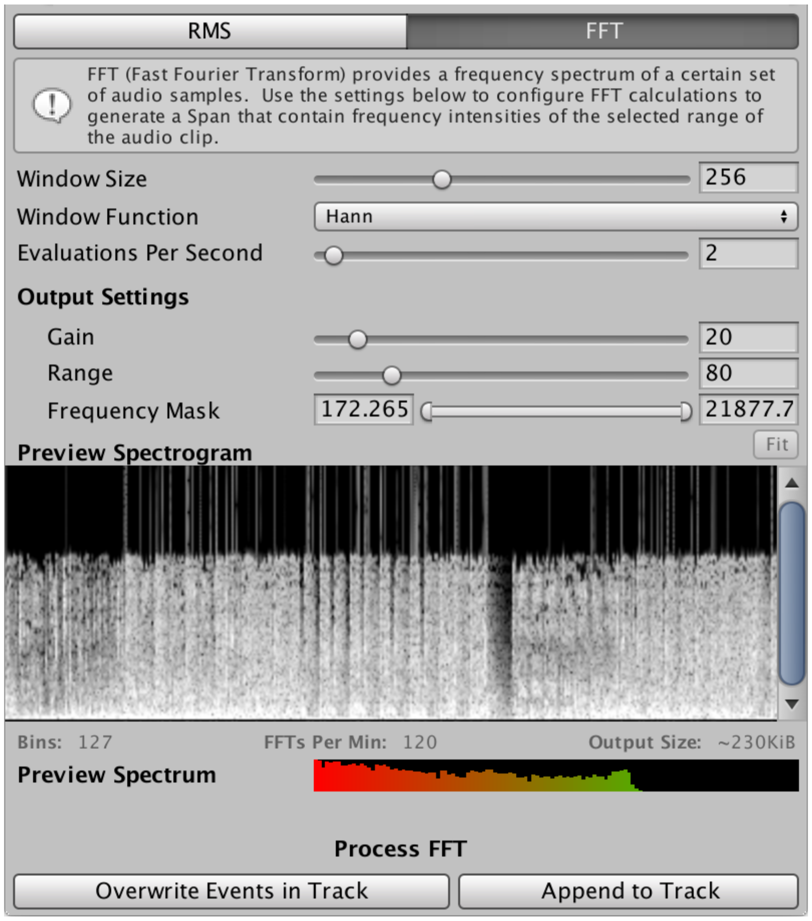
Configurable Fields

RMS Analysis has the following configurable fields that will help you customize your output:

- **Evaluation Frequency:** The number of data points evaluated in a given sample range. Increasing this value will reduce the number of data points to be calculated in the selected range (one of every 'n' available). The current zoom level of the Koreography Editor determines the number of samples to calculate per point.
    - *Samples Per Point:* [Non-Configurable] The number of audio samples to run RMS over for a single datapoint. This is the same number of audio samples used to create a single peak in the RMS waveform. The value can be changed by zooming in or out of the waveform in the Koreography Editor.
- **Output Settings:** Settings to adjust the output Koreography Events.
    - **Payload Type:** Set the Payload type for the output of the algorithm: either Curve or Float.
    - **Value Range:** The range of values for the final output. The default range of RMS is [0,1]. This setting will allow you to shift or otherwise stretch/compact that range to suit your needs.

FFT Analysis

This audio analysis method uses Fast Fourier Transform calculations to create Koreography Events with Spectrum Payloads. The FFT analyzes the audio using the configured settings and outputs a series of frequency spectra over time. The Spectrum Payload data can be used to create a spectrum visualizer, frequency-specific effects, and more. As with RMS analysis, results are most effective when applied to data containing very few “voices”, such as speech or stem files.



RMS | FFT

FFT (Fast Fourier Transform) provides a frequency spectrum of a certain set of audio samples. Use the settings below to configure FFT calculations to generate a Span that contain frequency intensities of the selected range of the audio clip.

Window Size: 256
Window Function: Hann
Evaluations Per Second: 2

Output Settings

Gain: 20
Range: 80
Frequency Mask: 172.265 – 21877.7

Preview Spectrogram

Bins: 127 FFTs Per Min: 120 Output Size: ~230KiB

Preview Spectrum

FFT Analysis Settings

Configurable Fields

FFT Analysis has the following configurable fields that will help you customize your output:

- **Window Size:** The number of samples to send to the FFT algorithm. The larger the window, the higher the frequency resolution of the output. However, larger windows generate more data and aren't as localized in time.
- **Window Function:** The windowing function to use during FFT processing. Different windows produce slightly different results. The Rectangular window is equivalent to no window at all and typically results in the largest amount of noise.
    - **Rectangular:** The Rectangular window function. This is equivalent to using no window and tends to produce noisy results.
    - **Hann:** The Hann window function. This is the default setting as it tends to produce results with lower amounts of noise.
    - **Hamming:** The Hamming window function. This is another commonly used window function. It tends to produce less noise than the Rectangular window and may be better than the Hann window with certain types of audio.

- **Evaluations Per Second:** The number of times the FFT should be applied for every second selected in the configured Sample Range. The calculations are evenly spread and applied windows may overlap.
- **Output Settings:** Settings to adjust the output Koreography Events.
    - **Gain:** The number of decibels by which to raise all FFT results. May bring out quieter peaks.
    - **Range:** The decibel range to consider to showcase results. A wider range may result in more visual noise, while a lower range may highlight dramatic changes.
    - **Frequency Mask:** The range of frequencies to select for export. Frequency resolution is dependent upon Window Size.

Preview Fields

In addition to configurable fields, the FFT Analysis window provides some preview fields to help visualize the expected output. The following fields are available:

- **Preview Spectrogram:** A spectrogram showing a sampling of frequencies evaluated over time given the current FFT settings. Each column of pixels is a representation of the linear frequency spectrum.
    - **Fit:** Whether or not the preview texture should be vertically stretched or squashed to fit the preview area.
    - **Bins:** The number of frequency bins that will be exported with the current settings.
    - **FFTs Per Min:** The number of FFTs analyzed per minute. Similar to BPM!
    - **Output Size:** The estimated size of the resulting payload data.
- **Preview Spectrum:** A preview of the spectrum at the location of the mouse when the mouse is over the Preview Spectrogram. Note that the number of bars represented may be smaller than the total number of available bins.

Note: The configured Sample Range is currently ignored for Preview Fields. The previews do not necessarily reflect output Koreography.

Keyboard Shortcuts

The Koreography Editor has support for many useful keyboard shortcuts to help enhance productivity. Most of the commands only work when the Waveform View has focus. See the following list for available commands:

Shortcut Key		Usage
Windows	Mac OSX	
A		Toggles **Select** mode for mouse interactions with Koreography Events
S		Toggles **Draw** mode for mouse interactions with Koreography Events
D		Toggles **Clone** mode for mouse interactions with Koreography Events
Z		Toggles **OneOff** Koreography Event generation
C		Toggles **Span** Koreoraphy Event generation

E *or* Enter *or* Return	E *or* ↲ *or* ↩	**Insert** new Koreography Event at playhead [during playback]
V		Toggles the **Visualizer** window visibility
Esc	⎋	**Focus** Waveform View if not already focused; if focused **clear** the active event selection
Shift	⇧	Inverts the **Snap To Beat** setting for some operations when held
Alt	⌥	Enables **Audio Scrubbing** when moving the mouse over the Waveform Display
Space		**Play/Pause** audio
Shift+Space	⇧Space	**Stop** audio
Left Arrow *or* Right Arrow	← *or* →	**Move the playhead** back/forward one measure
Shift+Left Arrow *Or* Shift+Right Arrow	⇧← *or* ⇧→	**Move the playhead** back/forward one beat
Down Arrow *or* Up Arrow	↓ *or* ↑	Decrease/Increase **playback speed** by 0.1x
Shift+Down Arrow *or* Shift+Up Arrow	⇧↓ *or* ⇧↑	Decrease/Increase **playback speed** by 0.01x
Backspace *or* Del	⌫ *or* ⌦	**Delete** selected Koreography Event(s)
Ctrl+A	⌘A	**Select All** Koreography Events
Ctrl+X	⌘X	**Cut** selected Koreography Event(s) to clipboard*
Ctrl+C	⌘C	**Copy** selected Koreography Event(s) to clipboard*
Ctrl+V	⌘V	**Paste** Koreography Event(s) from clipboard*
Ctrl+Shift+V	⇧⌘V	**Paste earliest Payload** from clipboard* into selected Koreography Event(s)
Ctrl+Z	⌘Z	**Undo**
Ctrl+Y	⇧⌘Z	**Redo**
Ctrl+Shift+K	⇧⌘K	**Open** the Koreography Editor
Alt+Shift+K	⌥⇧K	**Open** the MIDI Converter**

* The Koreography Editor maintains an internal clipboard that is separate from the system clipboard. Related operations will only work when the Waveform View has focus.

**The MIDI Converter is a feature that is specific to Koreographer Pro licenses only.

Saving Koreography Data Changes

In Unity, the typical save command (Ctrl+S on Windows, ⌘S on Mac) only saves the currently open scene(s). In order to save changes to Koreography, please use the *Save Project* option found within the main *File* menu. It should be noted that Unity will also automatically save changes to data when closed.

Runtime

The real power of Koreographer lies in the Runtime; the systems that bring everything together while your game or application is running. It brings tight synchronization and sends Koreography Events to systems/scripts configured to listen for them. The Runtime consists of three major pieces:

1. **Audio Player:** This is the system or set of scripts responsible for the playback of audio. Koreographer comes with several Audio Players.
2. **Koreographer (component):** This is the eponymous system in Koreographer responsible for sending out Koreography Events to listeners. It can also provide a programmatic interface to live Music Time (time in beats, rather than seconds).
3. **Audio Visor:** This script is responsible for Koreographer's tight synchronization. It "watches" the audio state of the Audio Player and feeds the resultant timing information to the Koreographer component. Koreographer's Audio Players all provide their own Audio Visors.

Combined with Koreography data, these pieces work together to produce the tight synchronization of music and gameplay/visuals/etc. The Audio Player will drive the audio, the Audio Visor communicates audio status to the Koreographer component, and the Koreographer component will trigger events based on loaded Koreography data.

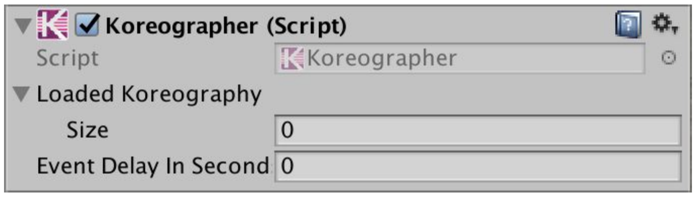
If configured properly, the Koreographer component maintains a special internal reference to the Audio Player in order to request information at runtime about playback state in order to enable Music Time API features. All Audio Players that come with Koreographer are configured to handle this.

This section will detail the various runtime components that make up the Koreographer Runtime system.

Koreographer

The Koreographer script is required in any scene in which you wish a system to receive Koreography Event notifications. Internally, the Koreographer class loosely follows the singleton design pattern: when a Koreographer component starts up, it sets itself as *the* Koreographer instance. This allows easy access to the Koreographer component API for the most common use cases.

That the Koreographer class only *loosely* follows the singleton pattern is important: you can easily run two or more Koreographer components in the same scene. Each will overwrite the static singleton "`Instance`" reference during initialization. This means that the static Koreographer class APIs will only reach the last Koreographer instance to be initialized. You can, however, use direct references to differentiate between Koreographer instances.



Koreographer (Script)

Script	Koreographer
Loaded Koreography	
Size	0
Event Delay In Second	0

The Koreographer component Inspector

Configurable Fields

The Koreographer component has the following configurable fields:

- **Loaded Koreography:** A list of Koreography data to load at startup. This is useful if your Audio Player does not load Koreography for you.
- **Event Delay In Seconds:** How many seconds to delay the triggering of Koreography Events from reported audio time. Cannot be less than zero.

Special Note About Event Delay

Some platforms, particularly certain mobile devices, can add upwards of several hundred milliseconds of latency between when the audio system sends audio data (samples) to the speakers and when audio is *actually* heard. All devices have such a latency, although some are worse than others. If you are targeting certain platforms (mobile phones or consoles [potentially connected to home theater systems], for example), you may want to consider adding a configuration step to your game in which you allow the player to compensate for this latency by adjusting this value.

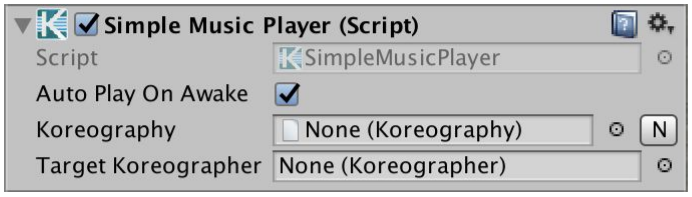
Players

While it is possible to integrate Koreographer into a pre-existing audio solution, several audio players are included by default to ease integration. These audio players enable both the Koreography Event triggering and Music Time API features. This section will detail the included Audio Players, as well as provide an overview of how to create a custom Audio Player.

Simple Music Player

The Simple Music Player component is the simplest of the included Audio Players. Instead of taking an AudioClip reference, it takes a single Koreography reference. At runtime, the Koreography is loaded into the static Koreographer Instance. Adding this component will also add an AudioSource component, which is used internally to control audio. The Simple Music Player uses the AudioClip reference in the referenced Koreography for audio playback.

While this is called the Simple *Music* Player it, like the AudioSource component it wraps, is capable of playing normal audio files as well.



Simple Music Player (Script)

Script	SimpleMusicPlayer
Auto Play On Awake	✓
Koreography	None (Koreography)
Target Koreographer	None (Koreographer)

The Simple Music Player component Inspector

Configurable Fields

The Simple Music Player has the following configurable fields:

- **Auto Play On Awake:** When checked, the Simple Music Player will automatically play the Koreography specified in the following field, if any.
- **Koreography:** If configured, the Simple Music Player will load this Koreography at startup.
- **Target Koreographer:** This field is optional. It allows you to specify a target Koreographer component to use for Koreography Event reporting and Music Time API support. If no Koreographer component is specified, the default singleton Koreographer component reference will be used.

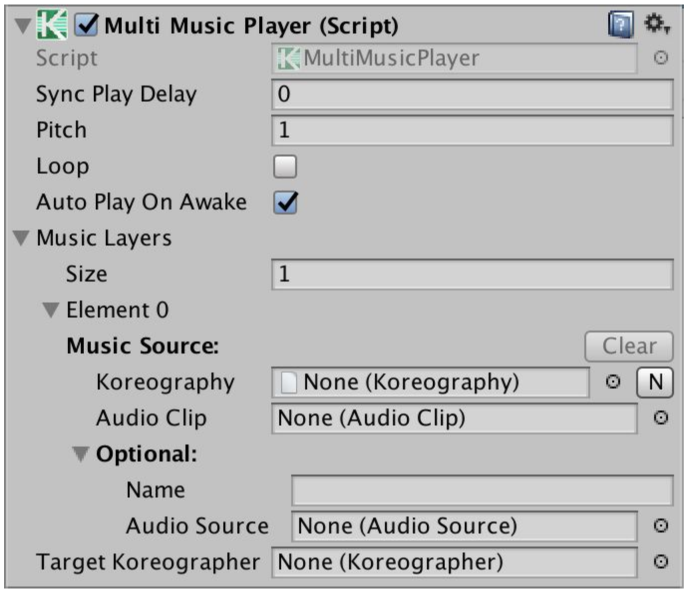
Some other settings, such as pitch and volume, can be configured on the AudioSource component that is added.

Multi Music Player

The Multi Music Player will *attempt* to play multiple synchronized layers of audio. These layers can be configured using raw AudioClip references or with Koreography references. The interface disallows setting both for a single layer.

Under certain conditions, the layers may not all start at the same time. This is usually due to heavy CPU load. Use the Sync Play Delay feature to schedule the playback of audio in a synchronized fashion.

Internally, each layer requires an AudioSource component for playback of the AudioClip. It is possible to specify a specific AudioSource component for a layer to use. If no AudioSource is specified, the system will add one to use automatically.


Multi Music Player (Script)
Script: MultiMusicPlayer
Sync Play Delay: 0
Pitch: 1
Loop: ☐
Auto Play On Awake: ✓
Music Layers
Size: 1
Element 0
Music Source:
Clear
Koreography: None (Koreography)
N
Audio Clip: None (Audio Clip)
Optional:
Name:
Audio Source: None (Audio Source)
Target Koreographer: None (Koreographer)

The Multi Music Player component Inspector

Configurable Fields

The Multi Music Player has the following configurable fields:

- **Sync Play Delay:** Will add a specific offset to the start of music playback. As this is passed to Unity's internal engine, it does a better job "guaranteeing" a synchronized startup across layers.

- **Pitch:** Adjusts the pitch of all layers simultaneously.
- **Loop:** If checked, the Multi Music Player will loop all the layers.
- **Auto Play On Awake:** When checked, the Multi Music Player will automatically play audio specified in the Music Layers. Otherwise, the audio and/or Koreography will simply be loaded and prepared for playback.
- *Music Layer Settings:*
    - *Music Source:* Where this layer will find audio data. The **Clear** button will reset these values to *None*.
        - **Koreography:** Koreography to load for this layer. The Koreography will be loaded into the static Koreographer Instance while the AudioClip in the Koreography will be used for playback.
        - **Audio Clip:** Don't load Koreography - simply play the AudioClip.
    - *Optional:* The settings in this section are optional.
        - **Name:** Adds a name to the layer. This is used to rename the Music Layer entry in the list in the Inspector. In the screenshot above, this value would replace "Element 0".
        - **Audio Source:** A reference to an AudioSource component that this Music Layer should use for playback.
- **Target Koreographer:** This field is optional. It allows you to specify a target Koreographer component to use for Koreography Event reporting and Music Time API support. If no Koreographer component is specified, the default singleton Koreographer component reference will be used.

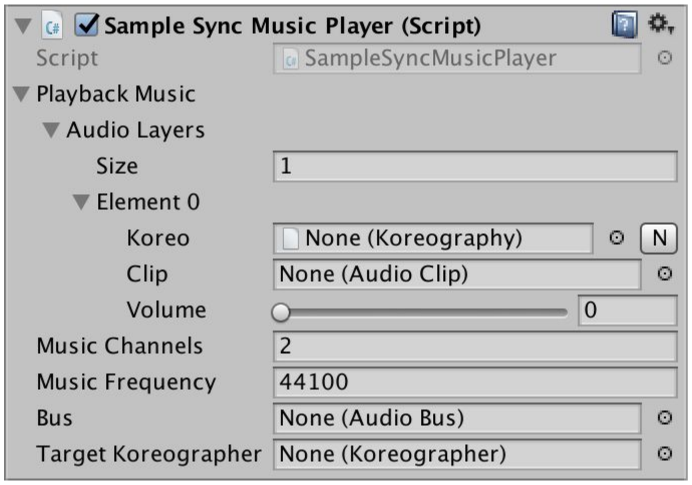
Pro Sample Sync Music Player [Experimental]

The Sample Sync Music Player is only available with Koreographer Pro. While the Multi Music Player *attempts* to play multiple audio layers in a synchronized fashion, the Sample Sync Music Player *guarantees* sample-specific synchronization. This player is **experimental**, meaning that it has not been rigorously tested (for the vast majority of use cases, the Multi Music Player has proven entirely sufficient) and is an extremely low priority in terms of support.

The Sample Sync Music Player works by loading all audio data for all layers in the mix and combining them *during* playback. This maintains the flexibility to change layer volume independently at the cost of [potentially quite a bit of] memory usage.

A separate Audio Bus component must be added for the system to work correctly. This Audio Bus component represents the connection between the Sample Sync Music Player and Unity's underlying audio system.

Note: The Audio Bus component is not related to Unity's Audio Mixer.


Sample Sync Music Player (Script)

Script	SampleSyncMusicPlayer
Playback Music	
Audio Layers	
Size	1
Element 0	
Koreo	None (Koreography)
Clip	None (Audio Clip)
Volume	0
Music Channels	2
Music Frequency	44100
Bus	None (Audio Bus)
Target Koreographer	None (Koreographer)

The Sample Sync Music Player component Inspector

Configurable Fields

The Sample Sync Music Player has the following configurable fields:

- **Playback Music (Audio Layers):** The audio, configured as layers, to playback simultaneously.
  - *Audio Layer Settings:*
    - **Koreo:** The Koreography to use for Koreography and audio playback.
    - **Clip:** The AudioClip to use for audio playback.
    - **Volume:** The initial volume for this Audio Layer.
- **Music Channels:** The number of channels in the configured audio. This must match the configuration of all AudioClips across all layers!
- **Music Frequency:** The sample rate of the configured audio. This must match the configuration of all AudioClips across all layers!
- **Bus:** An Audio Bus component reference. Simply add one to the same GameObject and drag it into this slot. The Audio Bus is what funnels audio data into the AudioSource that comes with the Sample Sync Music Player.
- **Target Koreographer:** This field is optional. It allows you to specify a target Koreographer component to use for Koreography Event reporting and Music Time API support. If no Koreographer component is specified, the default singleton Koreographer component reference will be used.

Note: If both the Koreo and Clip are set in the Audio Layer Settings, the Koreography is loaded into the Koreographer and the AudioClip is used for playback, *even if the configured Koreography does not match the AudioClip*.

Custom Players

If the three included Audio Players do not meet your needs, it is possible to create a custom Audio Player that hooks into the Koreographer Runtime system. A custom player essentially replaces one of the three main pieces that make up the system: the piece responsible for controlling audio playback.

The one requirement for successfully creating a custom player is the capability of tracking playback time. If your player uses Unity's built-in system you can probably use Koreographer's `AudioVisor` class to manage this. If you use another plugin or one of your own design, you may need to create this glue piece yourself.

In order to support certain features, the Koreographer component internally needs access to certain information usually provided by an Audio Player. Your custom player should implement the `IKoreographedPlayer` interface and, in your Awake() method, initialize the Koreographer component's `musicPlaybackController` field to itself. This interface is what allows Koreographer to get audio timing and state information, enabling access to the Music Time APIs.

Specific details regarding the `IKoreographedPlayer` interface and the Audio Visor can be found in the following subsections.

The IKoreographedPlayer Interface

The `IKoreographedPlayer` Interface is a class interface that defines several methods that you must implement. The Koreographer component has a reference to an object that implements this interface called `musicPlaybackController`. The Koreographer component uses the `musicPlaybackController` internally to provide access to the Music Time APIs.

The methods in this interface and their purpose are outlined below:

- **`int GetSampleTimeForClip(string clipName):`** Returns the current *sample time* of the clip in question. This should be zero if the clip in question is not loaded.
- **`int GetTotalSampleTimeForClip(string clipName):`** Returns the total *sample time* of the clip in question. This should be zero if the clip in question is not loaded.
- **`bool GetIsPlaying(string clipName):`** Returns whether or not the clip in question is currently playing.
- **`float GetPitch(string clipName):`** Returns the currently configured pitch of the Audio Player for the clip in question. This can also be thought of as "Playback Rate".
- **`string GetCurrentClipName():`** Returns the name of the currently playing audio clip. If no audio clip is loaded, returns the empty string (`""`).

The Role of the Audio Visor

The Audio Visor is the glue that holds the Audio Player and the Koreographer component together. The Audio Visor's sole responsibility is to call the Koreographer component's `ProcessKoreography` method each frame, which is what eventually leads to triggering Koreography Events.

The `AudioVisor` class that comes with Koreographer is custom designed to watch a Unity AudioSource. It handles looping, sudden jumps in audio position, and interpolates audio position when the game thread is running faster than the underlying audio thread (this is somewhat dependent upon the size of the Audio DSP Buffers).

If your Audio Player uses Unity's AudioSource for audio playback of pure AudioClips then you can easily make use of the built-in `AudioVisor`. If, however, your Audio Player combines data from AudioClips (as is done by the Sample Sync Music Player) or via some other method, you will need to write your own custom script to drive the Koreographer component. It is highly recommended to do this by extending the `VisorBase` abstract class, overloading its abstract methods. If you decide to write your own solution from scratch, please call `ProcessKoreography` once per frame, per audio clip/file.

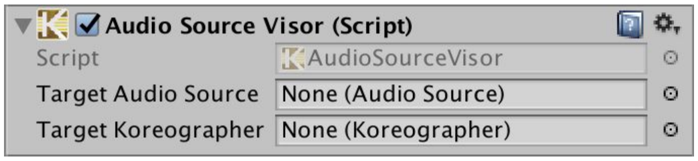
Note: `ProcessKoreography` takes a **`startTime`** and an **`endTime`** that defines a slice of time in **sample time** that equates to the current frame's `deltaTime`. The **`startTime`** you pass in for the current frame should be the previous frame's **`endTime + 1`**. Therefore, if you called `ProcessKoreography` with `startTime` and `endTime` values of `[128, 256]` last frame respectively, then this frame should have values of `[257, currentSampleTime]`. This is important because the Koreographer component checks both time bounds *inclusively*. If you send the same value twice and a Koreography Event happens to start or end on that sample position it will be triggered twice.

There are some tricky edge-cases when it comes to tracking audio position updates, particularly when it comes to looping or jumping audio positions which usually occurs in the middle of a specific visual frame, rather than synchronized to the frame's boundary. To properly handle this, Koreographer can internally split up a frame's `deltaTime` into slices. This is called the DeltaSlice. Please see the DeltaSlice documentation for more.

Audio Source Visor

Koreographer includes a very basic Audio Visor implementation, enabling quick access to runtime Koreography Event triggering. Specifically, the Audio Source Visor enables Koreography Event triggering based on the status of an AudioSource component. If an AudioSource is not explicitly specified, the Audio Source Visor will attempt to locate one on the same GameObject. As the Audio Source Visor does not implement the `IKoreographedPlayer` interface, this component will not enable access to Koreographer's Music Time API.

The Audio Source Visor will send timing information about the AudioClip loaded into the associated AudioSource component to either the singleton Koreographer component or, if set, a specific target Koreographer component. Koreography must be loaded in the Koreographer component used, either via its Loaded Koreography field or by the Koreographer component's script API.


Audio Source Visor (Script)

Script	AudioSourceVisor
Target Audio Source	None (Audio Source)
Target Koreographer	None (Koreographer)

The Audio Source Visor component Inspector

Configurable Fields

The Audio Source Visor has the following configurable field:

- **Target Audio Source:** If configured, the Audio Source Visor will use the specified AudioSource for playback status reporting. Otherwise, the Audio Source Visor will attempt to locate one on the same GameObject. If none is found, a warning will be printed to the console and the Audio Source Visor will disable itself.
- **Target Koreographer:** If configured, the Audio Source Visor will connect to the Koreographer component specified and use its Loaded Koreography for event triggering. Otherwise, the singleton Koreographer will be used.

Koreography Event Handling

Koreography Events are generated by Koreographer as the audio in your game or application plays. Koreographer sends these events to systems and scripts that are configured to listen for them. This is called Registration and there are two ways to register: **with** and **without** extra timing information. See the following subsections for more specifics!

Note: The code snippets in the following sections assume the use of Koreographer's namespaces.

Registering for an Event

Registering for simple Koreography Event notifications is very simple. The first thing to do is write a method in your script that Koreographer can call. This method must be of the form of the `KoreographyEventCallback` delegate:

```
public delegate void KoreographyEventCallback(KoreographyEvent koreoEvent);
```

Once you have defined your method, you must tell the Koreographer component instance that you want to receive notifications. Most of the time you can do this using the static singleton `Koreographer.Instance` reference. When you register you must tell the Koreographer component the Event ID for which you wish to receive Koreography Event notifications[2]:

```
Koreographer.Instance.RegisterForEvents("beatDrops", OnBeatDrops);
```

Once your method is registered, it will be called anytime the Koreographer component detects that a Koreography Event with the given Event ID occurs. In the example above, this means that a method "`OnBeatDrops`" would be called anytime a Koreography Event occurred in the Koreography Track with Event ID "beatDrops".

A more complete example might look like the following:

```
void Start()
{
    Koreographer.Instance.RegisterForEvents("beatDrops", OnBeatDrops);
}

void OnBeatDrops(KoreographyEvent evt)
{
    // Do something cool, inspect evt.Payload, etc!
}
```

[2] All examples provided are written in C#. Koreographer supports all scripting languages supported in Unity. Concepts, if not syntax, are portable.

Koreography Events called in this way occur early in the Unity Engine's main Update pass (within the Game Logic phase).

Registering for an Event with Time

Koreography Event notifications can optionally contain extra timing information. This timing information can be used to offset position calculations or animation playback, determine whether a Span Event has reached its end, and more. Registering for Koreography Events that come with this additional timing requires that you write a method in your script that conforms to the `KoreographyEventCallbackWithTime` delegate:

```
public delegate void KoreographyEventCallbackWithTime(KoreographyEvent koreoEvent, int
    sampleTime, int sampleDelta, DeltaSlice deltaSlice);
```

You will notice three extra parameters that do not appear in the plain `KoreographyEventCallback` delegate:

1. **`sampleTime`:** The current time for this Koreography Event.
2. **`sampleDelta`:** The number of samples that were played back since the previous frame. You can get the previous frame's `sampleTime` with (`sampleTime - sampleDelta`).
3. **`deltaSlice`:** Extra timing information required for simulation stability when the callback is called multiple times in a frame. See the Overview of the DeltaSlice section for more.

You can use these parameters in your callback to perform certain useful calculations.

Once you have defined your callback method, you must register it with a Koreographer component. This is most commonly done through the static singleton `Koreographer.Instance`:

```
Koreographer.Instance.RegisterForEventsWithTime("beatDrops", OnBeatDrops);
```

Once your method is registered, it will be called anytime the Koreographer component detects that a Koreography Event with the given Event ID occurs.

A more complete example might look like the following:

```
void Start()
{
    Koreographer.Instance.RegisterForEventsWithTime("beatDrops", OnBeatDrops);
}

void OnBeatDrops(KoreographyEvent evt, int sampleTime, int sampleDelta, DeltaSlice
    deltaSlice)
{
    /* Do something cool with the KoreographyEvent object and extra timing information!
       For example, you could use the sampleTime to retrieve a value from a Curve
       payload of a Span with evt.GetValueOfCurveAtTime. */
}
```

Koreography Events called in this way occur early in the Unity Engine's main Update pass (within the Game Logic phase).

Unregistering for Events

When you want a system or script to *stop* receiving Koreography Event notifications, you simply unregister for events. There are a few ways to do this, depending on your goals.

If you would like to unregister for any and all notifications from a Koreographer component, you can simply call `UnregisterForAllEvents` on that component (typically this is the static singleton `Koreographer.Instance`).

If, however, you wish to unregister a specific callback, you may call `UnregisterForEvents` and pass in the Event ID and callback method that you registered with.

Once a method or script is unregistered it will stop receiving notifications, even if they would otherwise occur.

Overview of the DeltaSlice

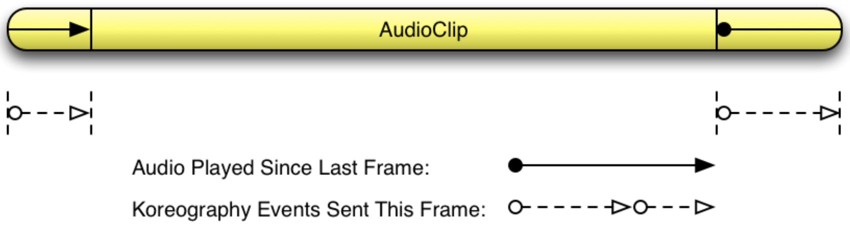
The DeltaSlice defines a slice - a *portion of* - the unscaled delta time of a given frame. It enables callback methods to properly calculate the specific timing of events with relation to the game frame. The DeltaSlice struct consists of the following two fields:

- **`deltaOffset`:** The starting point of the current unscaled delta of this slice. Range: [0,1].
- **`deltaLength`:** The length in seconds that this slice "consumes" of the unscaled delta.

The DeltaSlice is most commonly used in two scenarios:

1. **Audio Looping:** When an AudioClip loops, the time at which it does so has a very low probability of lining up precisely with game frame calculations. Therefore it is possible that a single frame will contain audio playback from the very end *and* very beginning of an AudioClip. Koreographer breaks this up into two separate Event notifications consisting of "the end" followed by "the beginning". As each of these takes up a *portion* of the overall `Time.unscaledDeltaTime` that constitutes the frame in question, just how much is used up by "the end" versus that used by "the beginning" is communicated in the DeltaSlice.
2. **Delayed Processing:** When Koreography Event processing is intentionally delayed, Koreographer maintains an AudioClip playback history that matches "time between frames" to "audio samples consumed between frames". This allows Koreographer to maintain synchronization in scenarios with variable frame rates, variable playback rates (pitch; speed), and intentional delay. In order to compensate for "past frame rates", Koreographer may send multiple events in a single processing pass. The amount of time of the current frame's `Time.unscaledDeltaTime` (in *most* cases) that an update represents is communicated in the DeltaSlice.

The following diagram shows a visual representation of the Audio Looping scenario outlined above:


AudioClip
Audio Played Since Last Frame:
Koreography Events Sent This Frame:

The Audio Looping scenario

Payload Handling

Payloads allow you to associate extra information about specific Koreography Events. You could add a number to indicate how intense a specific beat was in music, text for a lyric or subtitle, or a color for how a particular portion of your audio *feels*.

Accessing a Payload is as easy as checking the `Payload` property of the Koreography Event object you received in your callback. If this value is not `null`, the Koreography Event contains a Payload!

Payloads are implemented as simple classes that conform to the `IPayload` interface. By default, this is simply used as a classifier: there are no immediately useful runtime methods defined in the `IPayload` interface. Payload accessors are either defined in the `KoreographyEvent` class or individual Payload classes.

Type Verification

More often than not, it is important to verify the type of the Payload included in a Koreography Event. This can be accomplished in a number of ways:

- **`Has[Type]` Methods:** Each Payload type can be verified by calling `Has[Type]`. For example, if you want to check if a Koreography Event `evt` contained a `ColorPayload` reference you could check using `evt.HasColorPayload()`.
- **The `is` and `as` Operators:** The `Has[Type]` methods merely wrap a check using the `as` operator. For example, to check if a Koreography Event `evt` contained a `ColorPayload` reference you could check using: `if (evt.Payload is ColorPayload)`.
- **Raw Typecasting:** If you are *certain* about the underlying type, it is possible to access the Payload using a raw typecast. For example, to access a `ColorPayload` from a Koreography Event `evt`, you could use: `(ColorPayload)evt.Payload`.

Once you are certain of the type of the Payload you can access the important data contained within!

Accessor Overview

Accessing Payload data is simple. The Koreographer system provides two ways to access the data:

1. **Payload Class methods:** Once you have a reference to a Payload class instance (e.g. `ColorPayload`) you can call type-specific accessor methods on it. For the `ColorPayload` class this includes the `ColorVal` property.
2. **`KoreographyEvent` methods:** The `KoreographyEvent` class provides shortcut wrapper methods that simplify access. These are best used when you expect only a specific Payload type to arrive in your

callback. For example, you could use the `KoreographyEvent` `GetColorValue` method to retrieve the `Color` value stored in a `ColorPayload` instance.

It is often more advantageous to use direct Payload class methods through a local reference rather than to use the `KoreographyEvent` accessor methods as the latter can frequently lead to less performant code.

Performance Implications

The built-in accessors all verify the Payload type using the `as` operator prior to retrieving any data. This is not a free operation. If your callback expects to handle Koreography Events that have mixed Payload types, it is highly recommended that you use the `as` operator in your method, store the resulting pointer, and perform operations based on this local reference.

Direct Typecasting (`(ColorPayload)evt.Payload`, for example) has the same performance implications as the `as` operator above. Try to keep typecasting to a minimum!

Scripting Koreographer

Whether you are dealing with a precompiled version of Koreographer (scripts packaged as DLLs) or you have access to the source code (in its default folder structure), Koreographer's scripting API will work with any of the scripting languages supported by Unity. Koreographer itself was written entirely in C#.

Namespaces

Koreographer's scripting APIs are organized into a set of namespaces. This helps organize the codebase while reducing the chance that Koreographer's type names will collide with names in your own project. The namespaces Koreographer uses are as follows:

- **SonicBloom.Koreo:** Provides access to Koreographer's core classes and components.
- **SonicBloom.Koreo.Players:** Provides access to Koreographer's Music Player components and Audio Visor classes.
- **SonicBloom.Koreo.Demos:** Provides access to the Demo classes that Koreographer ships with.
- **SonicBloom.Koreo.EditorUI:** Namespace used for Editor-side functionality.
- **SonicBloom.Koreo.EditorUI.UnityTools:** Namespace used for Editor integration.
- Pro **SonicBloom.MIDI:** Provides Sonic Bloom's core MIDI type support.
- Pro **SonicBloom.MIDI.Objects:** Provides support for Sonic Bloom's MIDI object representation.

In order to access features of Koreographer, you must add a `using` statement to the top of your script. See:

```
// Compilers will now recognize Koreographer, Koreography, KoreographyEvent, etc.
using SonicBloom.Koreo;
```

Alternatively you can access classes directly without the `using` statement:

```
// Clear all event registrations from the singleton Koreographer.
SonicBloom.Koreo.Koreographer.Instance.ClearEventRegister();
```

Most IDEs (e.g. MonoDevelop and Visual Studio) will autocomplete the namespaces for you as you type.

Windows Platforms (Non-Desktop)

Koreographer has full support for non-desktop Windows platforms. These include:

- Windows Phone 8
- Windows Store, including:
  - Desktop
  - Mobile
  - Universal

Due to plugin enhancements made in Unity 5.0, enabling Koreographer support for these platforms may depend on your Unity version:

- **In Unity 4.5/4.6:** Please use the source code version of Koreographer to target these platforms.
- **In Unity 5.0+:** Please import the *`Plugins/Koreographer/WinRT/WinRTSupport.unitypackage`* file into your project.

Note: Upon import of the *WinRTSupport.unitypackage* in Unity 5.0~5.3, you may see a **`System.Reflection.TargetInvocationException`** in your console. This is a known issue and may be safely ignored: it only occurs on import and does not affect the build.

No special modifications to Koreographer are required.

Glossary of Terms

Definitions of commonly used terms.

- **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 formats that Unity can import are .aif, .wav, .mp3, and .ogg."
- **Audio Visor:** An Audio Visor is a system responsible for tracking the audio position of a given music or audio clip/file. Audio Visors report timing information to a Koreographer component which uses that information to identify potential Koreography Events to trigger.
- **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.
- **Koreographer:** Koreographer refers to both the Koreographer system as a whole and the runtime Koreographer component. The component version *is* the Choreographer: it takes the current audio point, queries the Koreography Track(s) for events at the given times, and sends "cues" (callbacks) to "actors" (systems/scripts) that are listening for "directions" (signals).
- **Koreography:** The Koreography associates a single AudioClip with one or more Koreography Tracks. It is the package of data that informs the Koreographer component about what events to generate while the associated AudioClip is playing. It also holds the Tempo Map information that enables the Koreographer system to provide a Music Time interface.
- **Koreography Track:** A Koreography Track groups 0 or more Koreography Events together and is identified with a *globally* non-unique string Event ID. When registering for events with the Koreographer component, this Event ID is used to associate the callback with Koreography Events triggered from the specific Koreography Track. Currently, an Event ID *is* (or tries to be) unique within a given Koreography asset.
- **Koreography Event:** A Koreography Event identifies a specific location in time of an AudioClip. Koreographer Events are considered either:
  - a OneOff (will only trigger once per playback) or
  - a Span (potentially triggers multiple times per playback).
- **Music/Audio Player:** These are the systems responsible for playing back music or audio files. They implement the `IKoreographedPlayer` interface and connect to the Koreographer component, enabling the Music Time API. Koreographer's built-in music players also manage Audio Visors for the audio they play back.
- **Payload:** Payloads are packages of data that are attached to Koreography Events and accessed via the `KoreographyEvent.Payload` property. When attached to an event that spans multiple samples the payload value will be displayed in the Koreography Editor spanning those values. It is possible to extend the system with custom payload types. Currently there are eight available options:
  - **No Payload:** No payload is associated with the Koreography Event. The `Payload` property will return `null`.
  - Pro **Asset:** An asset is associated with the Koreography Event. The `Payload` property will return an object of type `AssetPayload`, providing access to the `Asset` object.

    - Pro **Color:** A color is associated with the Koreography Event. The `Payload` property will return an object of type `ColorPayload`, providing access to the `Color` value.
    - **Curve:** A curve is associated with the Koreography Event. The `Payload` property will return an object of type `CurvePayload`, providing access to an `AnimationCurve` object. These payloads are frequently used to animate systems/logic/assets in the scene/game/application.
    - **Float:** A `float` value (number) is associated with the Koreography Event. The `Payload` property will return an object of type `FloatPayload`, providing access to the `float` value.
    - **Int:** An `int` value (whole number) is associated with the Koreography Event. The `Payload` property will return an object of type `IntPayload`, providing access to the `int` value.
    - Pro **Gradient:** A color gradient is associated with the Koreography Event. The `Payload` property will return an object of type `GradientPayload`, providing access to the `Gradient` object.
    - Pro **Spectrum:** A set of frequency spectra is associated with the Koreography Event. The `Payload` property will return an object of type `SpectrumPayload`, providing access to the `List` of `Spectrum` objects.
    - **Text:** Text is associated with the Koreography Event. The `Payload` property will return an object of type `TextPayload`, providing access to the `string` value.
- **Sample:** 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).