Koreographer Quick Start Guide

for v1.0.1

Table of Contents

Overview

To demonstrate using Koreographer in a scene, we will create a basic Koreographer setup in a barebones Unity scene using the least possible number of steps.

Minimum Required Elements

- The *Koreographer* component.
- A component that implements *KoreographerInterface* (examples include the provided *SimpleMusicPlayer* and *MultiMusicPlayer* components).
- At least one *AudioClip* (with Load Type **not** set to *Stream from disc*). This can be any audio clip.
- At least one *Koreography* asset that in turn contains...
- At least one *KoreographyTrack* asset that in turn has...
- One or more *KoreographyEvent*s.
- One or more *MonoBehaviour* components that have registered for the Event ID of the *KoreographyTrack*.

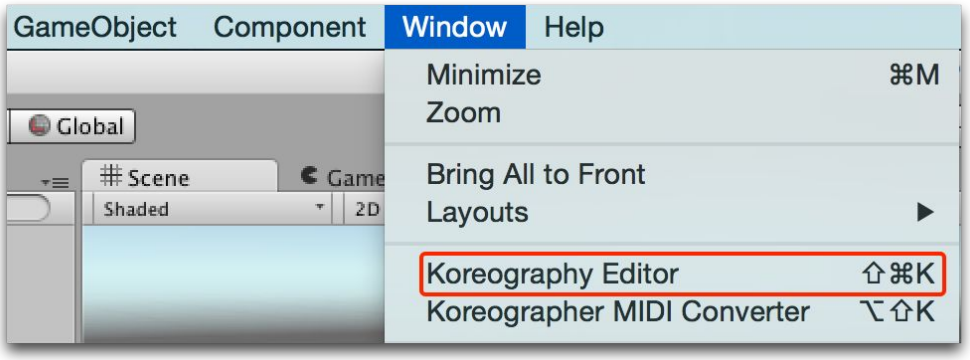
Outline of Instructions

1. Open an existing Unity project or create a new one.
2. Import the Koreographer Unity Package (generally this can be done via the Asset Store).
3. Open the Koreography Editor.
4. Create a *Koreography* asset file.
5. Associate the *Koreography* with an *AudioClip*.
6. Create a new *KoreographyTrack*.
7. Define the Event ID for the *KoreographyTrack*.
8. Create one or more *KoreographyEvent*s.
9. Create a *HelloKoreographer* MonoBehaviour.
10. Setup and test a Unity scene with the following components:
    a. *Koreographer*
    b. *SimpleMusicPlayer*
    c. *HelloKoreographer*

Detailed Walkthrough

Opening and Viewing the Koreography Editor

1. Open Unity and create a new Scene in either a new or existing project.
2. Import the Koreographer Unity Package (generally this can be done via the Asset Store).
3. Upon importing the Koreographer Unity Package, a **Koreography Editor** menu entry is added to the *Window* menu. Select that menu item to open the Koreography Editor.



The Koreography Editor can be opened from the Window menu

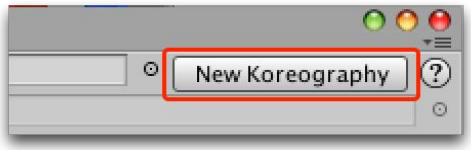
Koreography Editor

The Koreography Editor is the main tool used to create and edit:

- **Koreography:** An asset that associates an *AudioClip* with a group of *KoreographyTrack*s and a Tempo Map (defined through Tempo Sections).
- **KoreographyTracks:** A sequence of *KoreographyEvent*s defined along the audio timeline.
- **KoreographyEvent:** An event, either instantaneous (OneOff) or covering a span of time (Span), with either no payload or a payload consisting of a color, curve, float (number), int (whole number), gradient, or text (string) value.

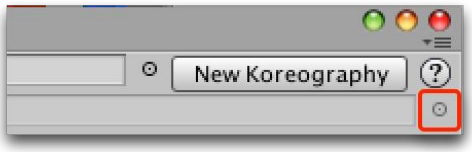
Create and Configure Koreography

4. Now that the Koreography Editor is open, we must create our first *Koreography* asset. At the upper-right hand corner of the editor window are buttons that allow you to either load an existing *Koreography* asset or create a new one.



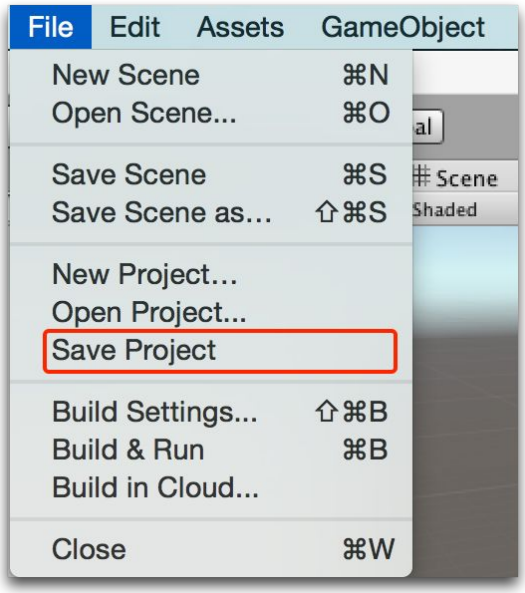
Create and load a new *Koreography* asset with the **New Koreography** button

   Click the **New Koreography** button, name the new asset file **MyFirstKoreography**, and save the file in the Assets folder of your Unity project.
5. Now that we have a *Koreography* asset loaded, we must associate an *AudioClip* with it. Select the Unity load widget "⊙" at the right-hand side of the *AudioClip* field and select your audio.



Use the Unity load widget "⊙" to load an *AudioClip*

   After loading, save the asset by going selecting "Save Project" from the File menu.



Save changes to Asset files in Unity with the **Save Project** menu entry

Create and Configure a KoreographyTrack

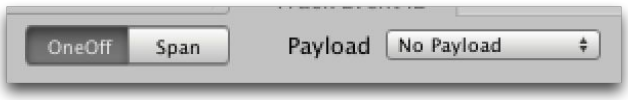
6. Now that we have a *Koreography* asset with an associated *AudioClip*, we must create a *KoreographyTrack*, which will contain our events. To do so, select the **New** Track button at the right side of the Koreography Editor window.

Create and Load a new *KoreographyTrack* asset with the **New** button

   Save the file as **MyFirstKoreographyTrack**.
7. When we create the *KoreographyTrack*, the Event ID is set to the name of the asset (a non-unique string ID). The Event ID is the key that classes use to register for events from *Koreographer* at runtime. Change the ID from the name of the file to **TestEventID**.

Create and Configure KoreographyEvents

Now we have all of the pre-requisite elements for creating *KoreographyEvent*s. Before we do so, we will configure the **structure** of the events we wish to create. To configure the Event Types and Payload Types, look at the right-hand side of the screen just above the waveform.



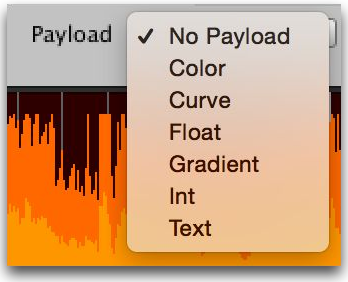
Event Type and Payload Type settings for newly created *KoreographyEvent*s

Event Types

- **OneOff:** If the **OneOff** button is depressed during *KoreographyEvent* creation, OneOff events are created.

- **Span:** If the **Span** button is depressed, events will span a range of time along the audio timeline. The event is started when an event creation key is pressed and ends when the depressed key is released.

Payload Types



Built-in Payload Types

- **No Payload:** This event is essentially a non-descript message; a trigger.
- **Color:** This event carries a *Color* value within it that can be retrieved by event subscribers.
- **Curve:** Same as the *Color* except that it carries an *AnimationCurve* object.
- **Float:** Same as the *Color* except that it carries a number.
- **Gradient:** Same as the *Color* except that it carries a *Gradient* object.
- **Int:** Same as the *Color* except that it carries a whole number.
- **Text:** Same as the *Color* except that it carries text.

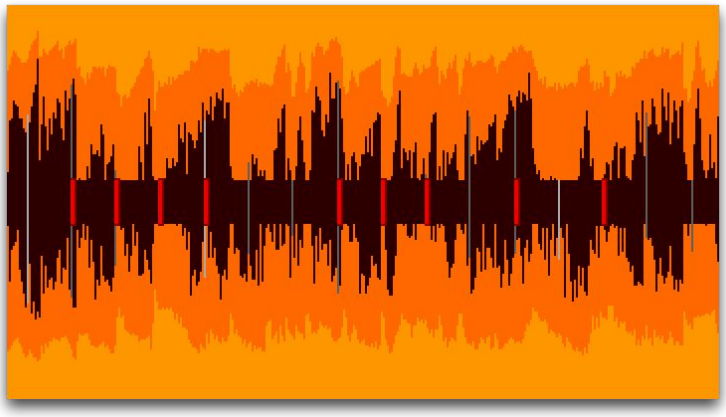
*KoreographyEvent*s can be created with either the **Draw Tool** or the keyboard during playback. For this guide we will add them with the keyboard during playback.

8. To create events with the keyboard we must play the *AudioClip* using the music player buttons on the left hand side of the screen.

The music player buttons

Once the audio is playing, we press **E**, **Return**, or **Enter** along with the point in the *AudioClip* where we would like to generate an event. *KoreographyEvent*s are generated using the event settings just above the Waveform Display and will display as vertical red bars for *OneOff* events or rectangular sections for *Spans*.

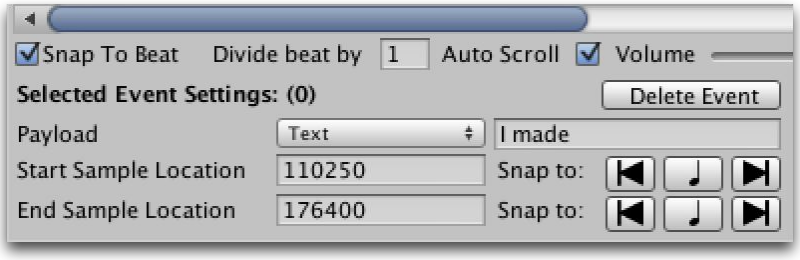
> **Note:** Just below the bottom-left corner of the Waveform Display are settings for ***Snap to Beat***. This enables quantization with the music and is on by default. Uncheck the box to allow for free-form music markup.
>
> With ***Snap to Beat*** checked, the Koreography Editor will use the information in the *Tempo Section Settings* to determine where beats fall within the music. The beat positions are depicted as vertical white lines behind the waveform. Adjust the tempo with the *Tempo Section Settings* tools to align the the beats to your music.



OneOff events

Span events

Payloads of selected events can be manually altered or tweaked after creation in the lower-left hand corner of the Koreography Editor. You can always view an event's *payload*, or lack thereof, by selecting the event and referring to the **Selected Event Settings** in the lower-left hand corner.



Example Text Payload settings

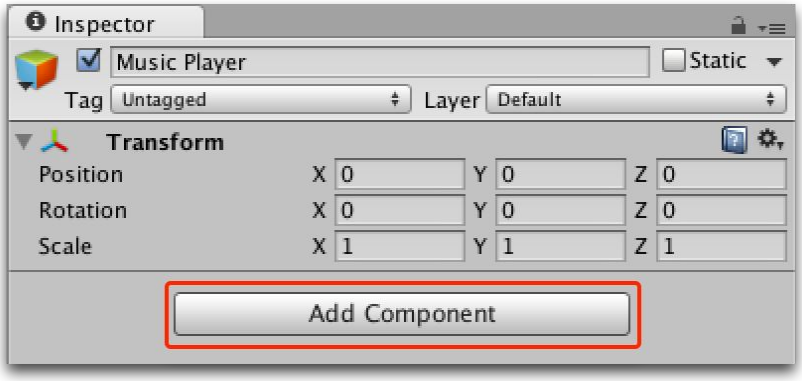
For more detail on event types and payload configurations, see the **Koreographer User's Guide** that accompanies this document.

Setting up Koreographer in the Unity Scene

9. Now that we have created a *Koreography* asset and associated an *AudioClip*, a *KoreographyTrack*, and *KoreographyEvents* with it, we can move on to setting up our scene. Our first step is to create a *GameObject* to hold our *SimpleMusicPlayer* and *Koreographer*.
    a. Create an empty *GameObject* for this scene and rename it *Music Player*.

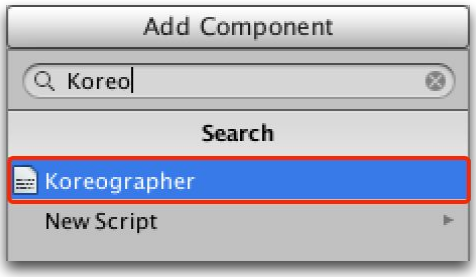
Select **Create Empty** from the GameObject menu

    b. Select the *GameObject* in the Hierarchy and select **Add Component** in the Inspector.



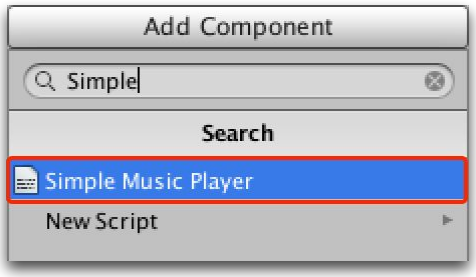
The **Add Component** button

    c. Type **Koreographer** into the search fieldand select the *Koreographer* script to add it to this *GameObject*.



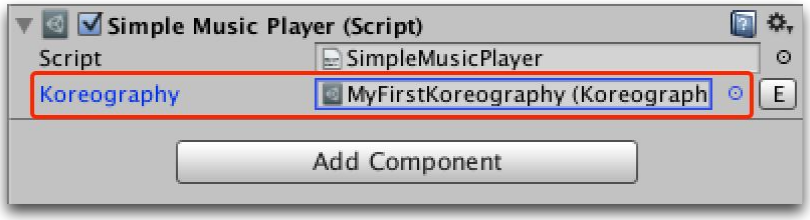
Add the *Koreographer* component

d. We will now repeat the last two steps to add the *SimpleMusicPlayer* component to this object. Adding the *SimpleMusicPlayer* script will also add the required *AudioSource* component.



Add the *Simple Music Player* component

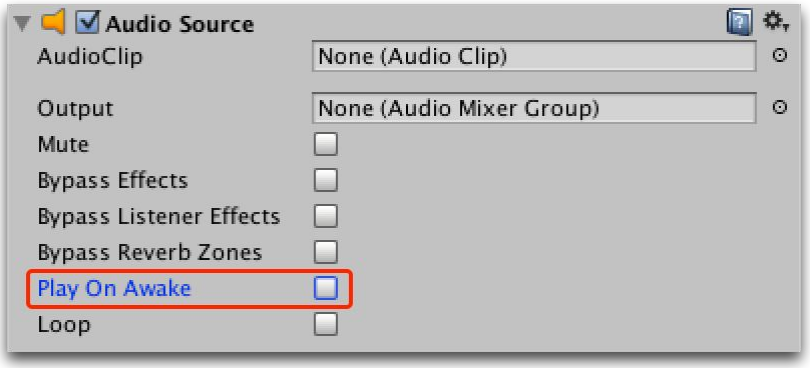
10. Now that we have a *GameObject* in the scene setup with the two necessary classes for tracking and firing events, we need to assign the *Koreography* that we configured. Assigning the *Koreography* asset will include the *KoreographyTrack*s, the *KoreographyEvent*s, and the *AudioClip* we assigned to it.
    a. We need to assign the **MyFirstKoreography** asset to the *SimpleMusicPlayer* component. This can be done by dragging our **MyFirstKoreography** asset into the empty form field or by selecting the Unity load widget "⊙" on the right side of the empty form field, selecting the Assets tab in the resulting popup window, and selecting the correct asset.



Setting the Koreography field of the *Simple Music Player*

> **Note:** There is no need to set the *AudioClip* on the *AudioSource* component. The *SimpleMusicPlayer* handles that for us on play!

    b. Our last step in setting up our *Koreographer* object in the scene is to uncheck the *AudioSource* **Play On Awake** property.

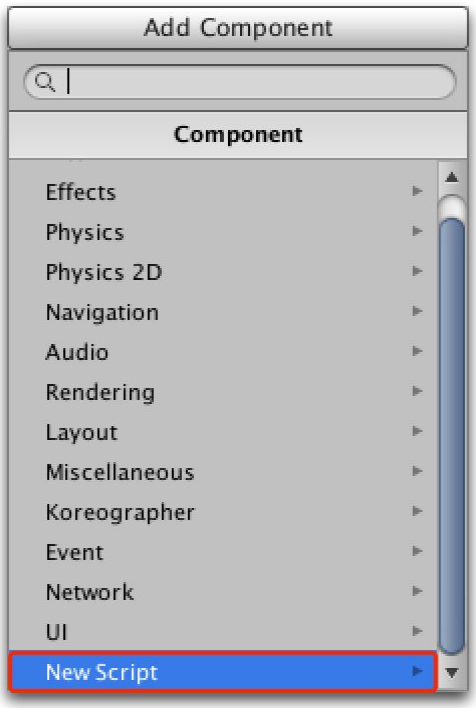


Uncheck **Play On Awake**

Create a Component that Handles KoreographyEvents

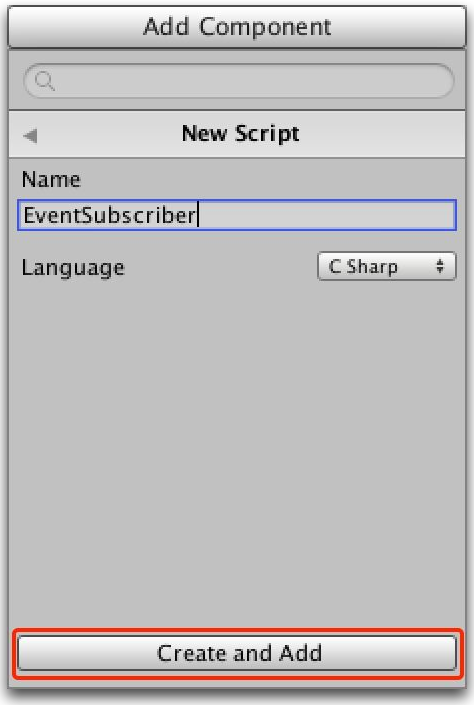
We now have a fully fledged *Koreographer* object ready to track and fire events. We just need a *GameObject* in the scene with the ability to subscribe to these events. To add this ability we will need to create a new component.

11. Create another empty *GameObject* in the scene and name it **TestEventSubscriber**.
12. Add a new C# script component to the newly created *GameObject*. To do this:
    a. Select **Add Component** under the **TestEventSubscriber** properties in the Unity Inspector and select the **New Script** menu item.



Select **New Script**

    b. Name the new script **EventSubscriber** and *switch the language to C#*. Confirm these settings and create/add the script to the **TestEventSubscriber** object by selecting **Create and Add**.



Create and Add the *EventSubscriber* script

You will see the *EventSubscriber* script now attached to the *GameObject* as well as in Unity's Project tab.

13. We now need to edit the script so that it can respond to events from our *Koreography*. Double-click it in either location and the script will open in your preferred script editor (this may be MonoDevelop or Visual Studio).
    a. At first, your script will be blank like so.

```
using UnityEngine;
using System.Collections;

public class EventSubscriber : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

    b. Subscribing to *Koreographer* for events is simple: we access the singleton instance (fancy name for a global object) of *Koreographer* and tell it what events we want to subscribe for.
    c. There are two things we want to consider when subscribing to events:
        - the Event ID and
        - the Callback Function to trigger when an event is fired.

The Event ID

This Event ID is what is set when creating the *KoreographyTrack* itself as either the name of the *KoreograpyTrack* file (default when first created) or to a custom name after creation. For our purposes, we set this to **TestEventID**. The Event ID allows us to create many distinct tracks of Koreography that can be triggered at the same time.

Callback Functions

Callback functions rely on function delegates. A function delegate defines the structure of a callback: its parameters and their types; its signature! We will create a function that uses the same structure as the Koreographer system's delegates. When we register this function with *Koreographer*, we will also register the Event ID.

Koreographer defines two function delegates we can use when creating callback functions:

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

The first delegate function *KoreographyEventCallback* defines a function with a return type of void and a single parameter of *KoreographyEvent*. This is used for when you only wish to receive the event object itself.

The second delegate function *KoreographyEventCallbackWithTime* defines a function with a return type of void and three parameters of *KoreographyEvent*, two *int*s (*sampleTime* and *sampleDelta*), and a *DeltaSlice* object. This is used for when you wish to receive the event and some information about where in the track the event occurred. The *sampleTime* parameter is the current time of the audio in samples at the moment the *KoreographyEvent* was sent while *sampleDelta* will be the total number of samples since the last frame (similar to how *Time.deltaTime* provides the canonical time since the previous frame). For more information on the *DeltaSlice* object, see the *Koreographer User's Guide*.

14. In order to use the Koreographer API we will need to add the namespace to the script. Add "`using SonicBloom.Koreo;`" beneath the "`using UnityEngine;`" directive:

```
using UnityEngine;
using SonicBloom.Koreo;
```

15. Create a single function that matches the first delegate (only a single parameter of *KoreographyEvent*). In the body of the function we will add a Debug statement to print a message about receiving an event. As you can see below, the structure of the function *FireEventDebugLog* matches that of the *KoreographyEventCallback* function delegate.

```
void FireEventDebugLog(KoreographyEvent koreoEvent)
{
    Debug.Log("Koreography Event Fired.");
}
```

16. Our last task before we test will be to register the Event ID and the callback function *FireEventDebugLog*. Registering for events is done by calling the *Koreographer* singleton's *RegisterForEvents* function. This is done once per instance of the object, so a good place to put it would be in the *Start* function.

```
// Use this for initialization
void Start()
{
    Koreographer.Instance.RegisterForEvents("TestEventID", FireEventDebugLog);
}
```

The Completed Component Script

The completed version of the *EventSubscriber* script will look like this. Save the file and close out your script editor.

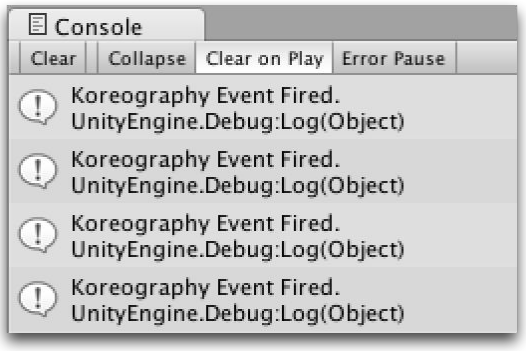
```
using UnityEngine;
using SonicBloom.Koreo;

public class EventSubscriber : MonoBehaviour
{
    // Use this for initialization
    void Start()
    {
        Koreographer.Instance.RegisterForEvents("TestEventID", FireEventDebugLog);
    }

    void FireEventDebugLog(KoreographyEvent koreoEvent)
    {
        Debug.Log("Koreography Event Fired.");
    }
}
```

Running and Testing the Koreographer Functionality

17. Now that we are back in Unity, we can test to make sure our events are being received. Press the Play button.

Unity's **Play Scene** button

As the audio plays and events are fired you should see log statements appear in the console from the *TestEventSubscriber*.



Watch the console for the Debug Output!