Koreographer

How-to Sample Project Manual 0.0.8

Jeff Campbell
Eric Robinson

Koreographer How-To Sample Project

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* (the provided *SimpleMusicPlayer* and *MusicPlayer* components do so).
- 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 referenced by the *KoreographyTrack*.

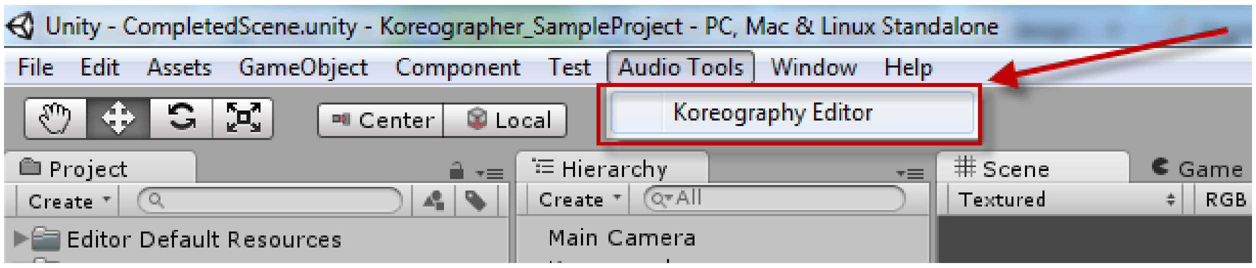
Outline of Instructions

1. Open an existing Unity project or create a new one.
2. Import the Koreographer Unity Package.
3. Open the Koreography Editor.
4. Create a *Koreography* file.
5. Associate the *Koreography* with an *AudioClip*.
6. Create a new *KoreographyTrack*.
7. Create one or more *KoreographyEvent*s.
8. Define the Event ID for the *KoreographyTrack*s.
9. Create our *HelloWorldKoreographerTester* class (MonoBehaviour-derived).
10. Setup our Unity scene.
11. Test our HelloWorld setup.

Steps

Opening and Viewing the KoreographyEditor

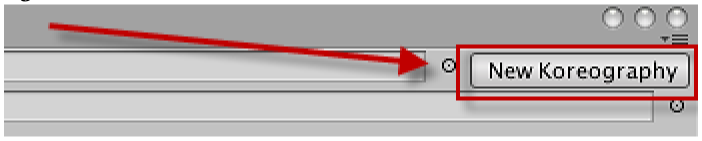
1. Open Unity and create and save a new Scene titled “Hello World Koreographer” in either a new or existing project.
2. Import the Koreographer Unity Package.
3. Upon importing the Koreographer Unity Package, a new menu option will become available to you titled **Audio Tools**. Underneath the *Audio Tools* menu bar, there is a single entry of Koreography Editor; select and open that menu item.


Unity - CompletedScene.unity - Koreographer_SampleProject - PC, Mac & Linux Standalone
File Edit Assets GameObject Component Test Audio Tools Window Help
Koreography Editor
Center Local
Project
Create
Editor Default Resources
Hierarchy
Create
All
Main Camera
Scene
Game
Textured
RGB

4. The Koreography Editor is the main tool used to create:
   - *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 curve, float (number), or text value.

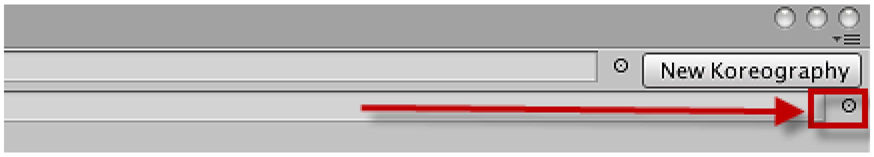
Create and Configure a Koreography

5. Now that the Koreography Editor is open, we must create our first *Koreography* asset. At the upper-right hand corner of the editor window, there is the option to load an existing *Koreography* asset or create a new one.



Select **New Koreography**, name the asset file as HelloWorldKoreography and save the file in the Assets folder of your Unity project.

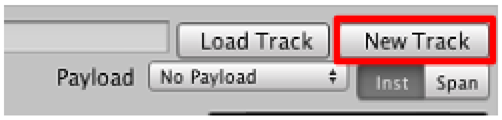
6. Now that we have a *Koreography* asset and it’s loaded, we must select an *AudioClip* to associate with it. Select the Unity load widget (small circle) at the right-hand side of the *AudioClip* form field and select your music.



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

Create and Configure a KoreographyTrack

7. Now that we have a *Koreography* asset and associated *AudioClip*, we must create a *KoreographyTrack*, which will contain our events. To do so, select the **New Track** button at the bottom-right corner of the *KoreographyEditor* window.

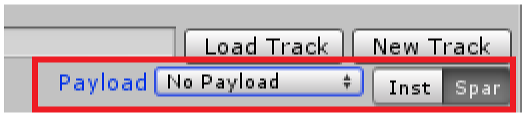


Save the file as **HelloWorldKoreographyTrack**.

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

Create and Configure KoreographyEvents

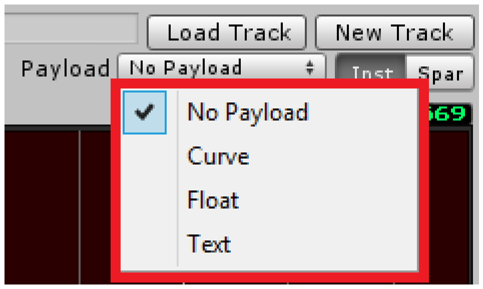
9. Now we have all of the pre-requisite elements for creating *KoreographyEvent*s. Before we do so, we will configure the **structure** of events we wish to create. To configure the event and payload types, look at the right-hand side of the screen just above the waveform.



9.1. *Event Types*

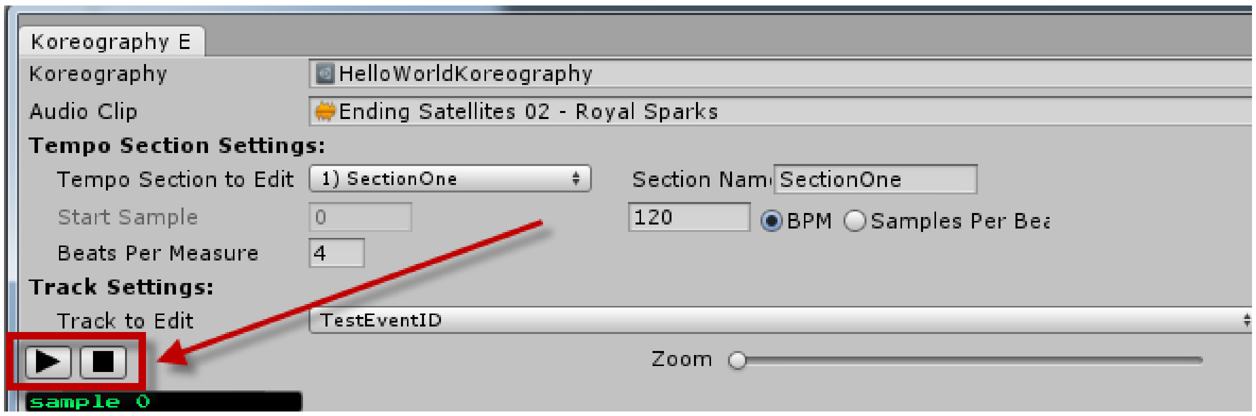
- *OneOff*: If the **OneOff** button is depressed, OneOff events are created while the keyboard is depressed. If an event creation key [**E**, **Enter**, or **Return**] is held down, multiple OneOff events will be generated.
- *Span*: If the **Span** button is depressed, events will span a range of time along the audio timeline. The event is started when an event creation key is pressed and ends when the depressed key is released.

9.2. *Payload Types*



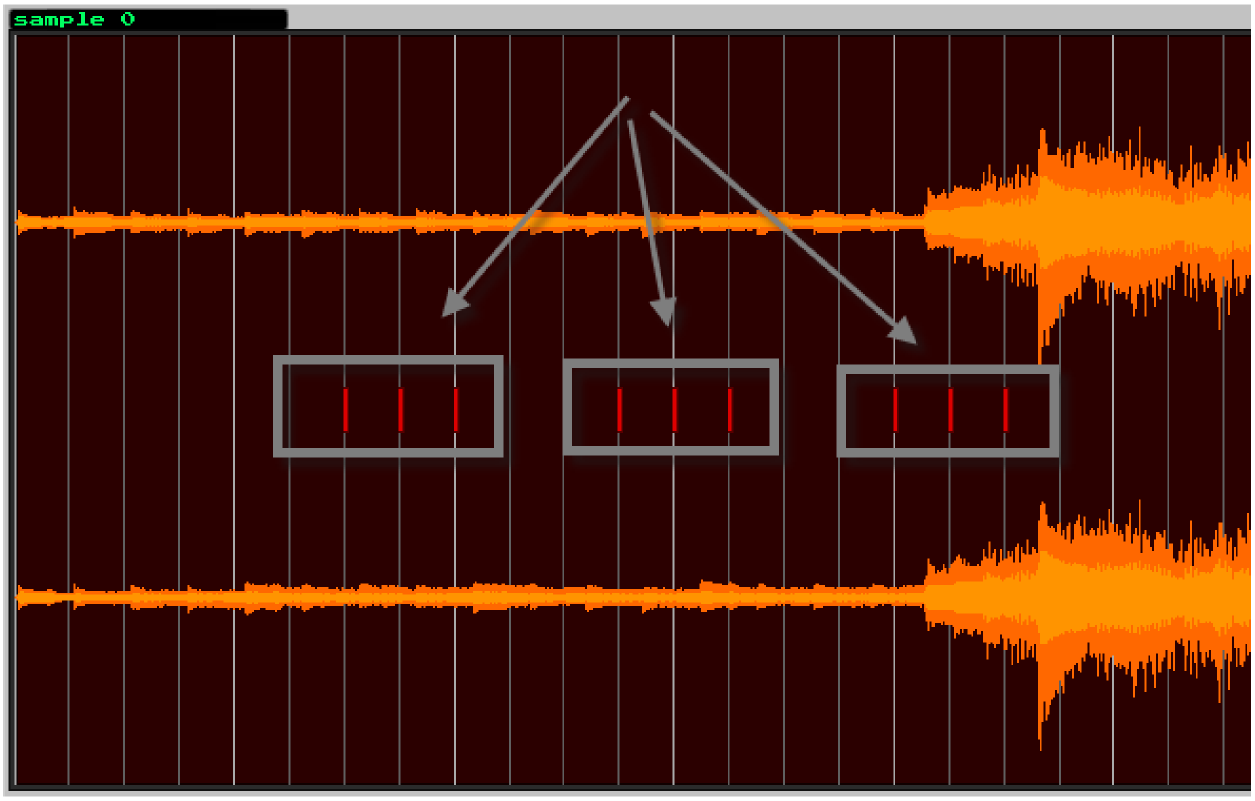
- *No Payload*: This event is essentially a non-descript message; a trigger.
- *Curve*: This event carries an *AnimationCurve* within it that can be retrieved by event subscribers.
- *Float*: Same as the *Curve* except that it carries a number.
- *Text*: Same as the *Curve* except that it carries text.

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

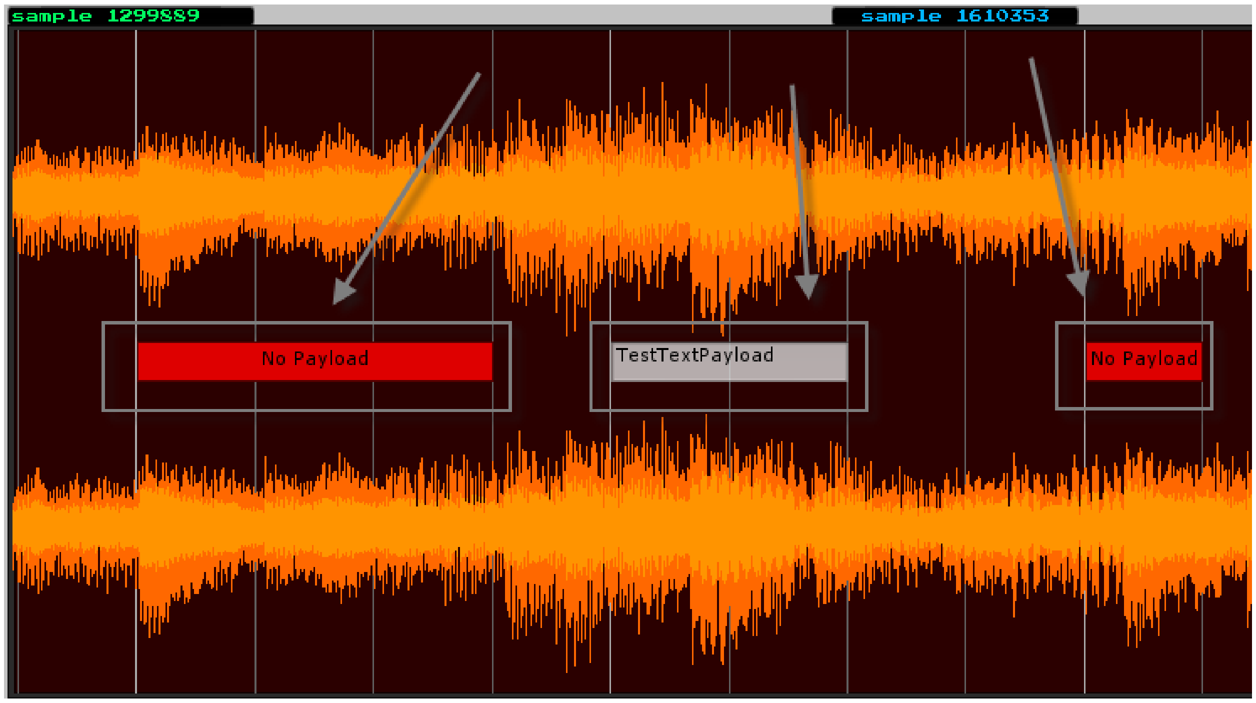

Koreography E
Koreography HelloWorldKoreography
Audio Clip Ending Satellites 02 - Royal Sparks
Tempo Section Settings:
Tempo Section to Edit 1) SectionOne
Section Nam SectionOne
Start Sample 0
120 ● BPM ○ Samples Per Bea
Beats Per Measure 4
Track Settings:
Track to Edit TestEventID
Zoom
sample 0

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. Events are generated using the event settings just above the upper-right hand corner of the Waveform Display and will display as red blips for *OneOff* events or rectangle sections for *Spans*.

> **Aside:** 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.

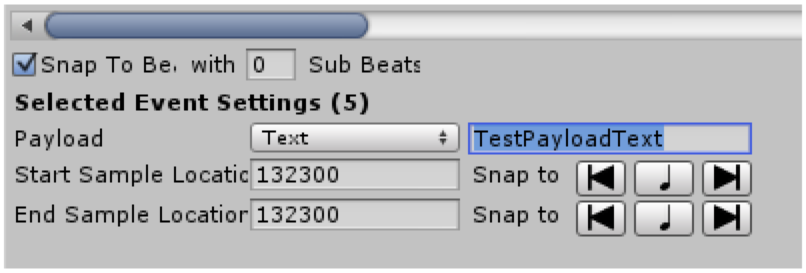


OneOff Events



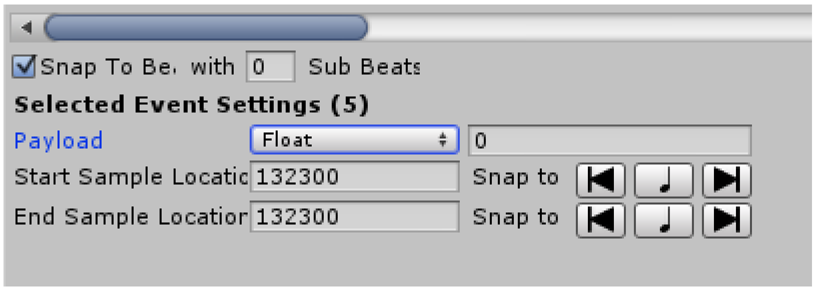
Span Events

Payloads 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.



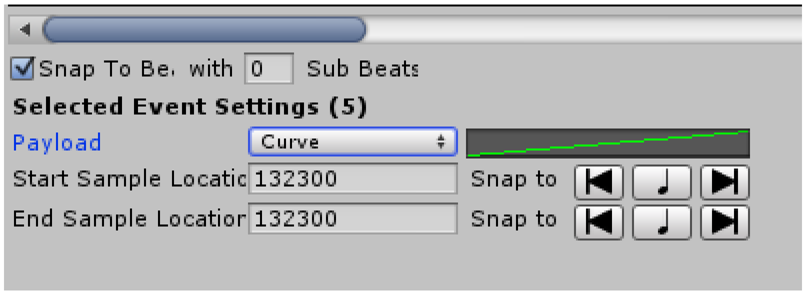
Snap To Be, with 0 Sub Beats
Selected Event Settings (5)
Payload Text TestPayloadText
Start Sample Locatic 132300 Snap to
End Sample Locatior 132300 Snap to

Example Text Payload Settings



Snap To Be, with 0 Sub Beats
Selected Event Settings (5)
Payload Float 0
Start Sample Locatic 132300 Snap to
End Sample Locatior 132300 Snap to

Example Float Payload Settings



Snap To Be, with 0 Sub Beats
Selected Event Settings (5)
Payload Curve
Start Sample Locatic 132300 Snap to
End Sample Locatior 132300 Snap to

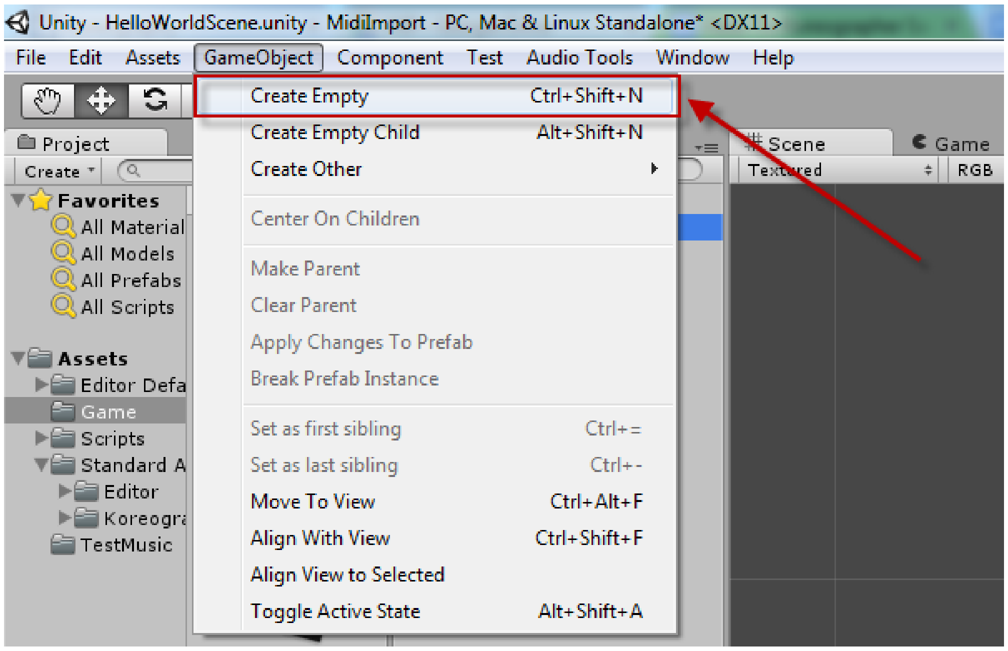
Example Curve Payload Settings

For more detail on event types and payload configurations, see the appendix.

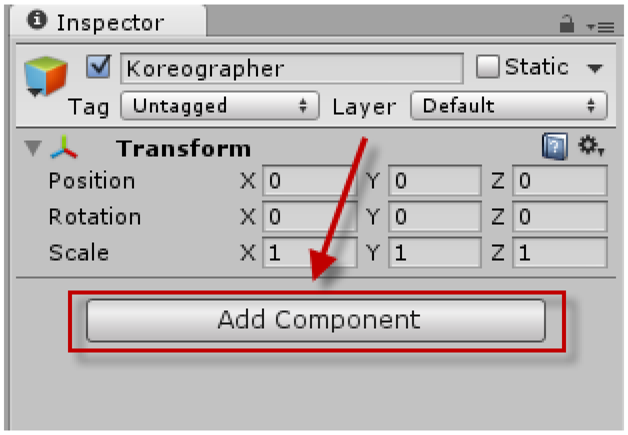
Setting up the Unity Scene

11. 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*.

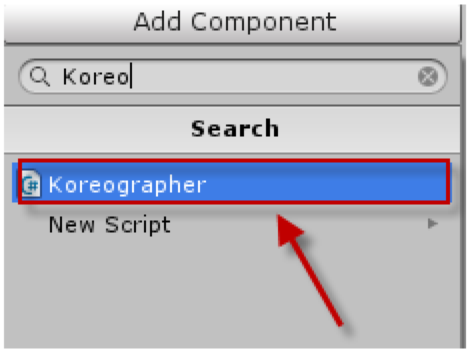
11.1. Create an empty GameObject for this scene and rename it *Koreographer*.


Unity - HelloWorldScene.unity - MidiImport - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Test Audio Tools Window Help
Create Empty Ctrl+Shift+N
Create Empty Child Alt+Shift+N
Create Other
Center On Children
Make Parent
Clear Parent
Apply Changes To Prefab
Break Prefab Instance
Set as first sibling Ctrl+=
Set as last sibling Ctrl+-
Move To View Ctrl+Alt+F
Align With View Ctrl+Shift+F
Align View to Selected
Toggle Active State Alt+Shift+A
Project
Favorites
All Material
All Models
All Prefabs
All Scripts
Assets
Editor Defa
Game
Scripts
Standard A
Editor
Koreogra
TestMusic
Scene
Game
Textured
RGB

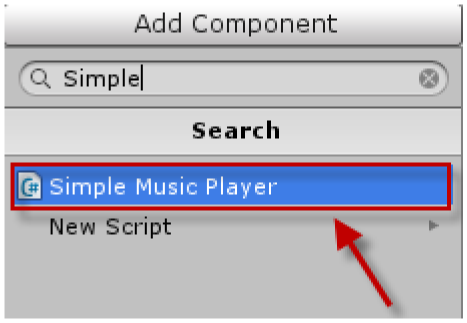
11.2. Select the game object in the Hierarchy and select **Add Component** in the Inspector.



11.3. Type in **Koreographer** and select the *Koreographer* script to add to this GameObject.

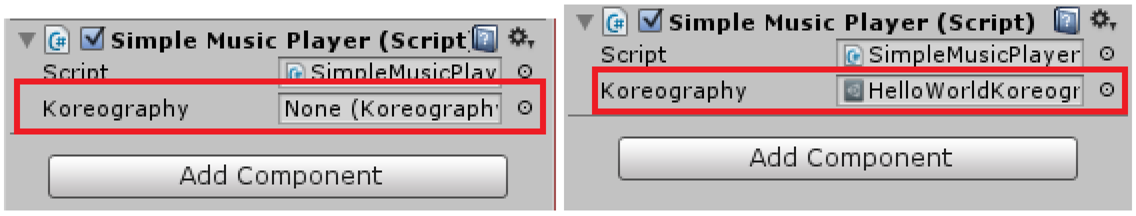


11.4. We will now repeat the last two steps to add the *SimpleMusicPlayer* script to this object. Adding the *SimpleMusicPlayer* script will also add the required *AudioSource* component.

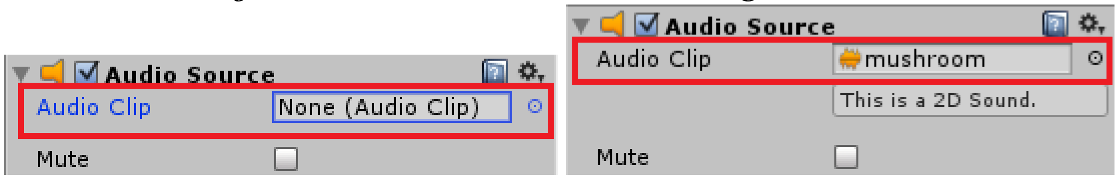


12. Now that we have a GameObject in the scene setup with the two necessary classes for tracking and firing events, we need to assign them the *Koreography* (and implicitly *KoreographyTracks* and *KoreographyEvents* that are associated with it) as well as the *AudioClip* it will be monitoring.

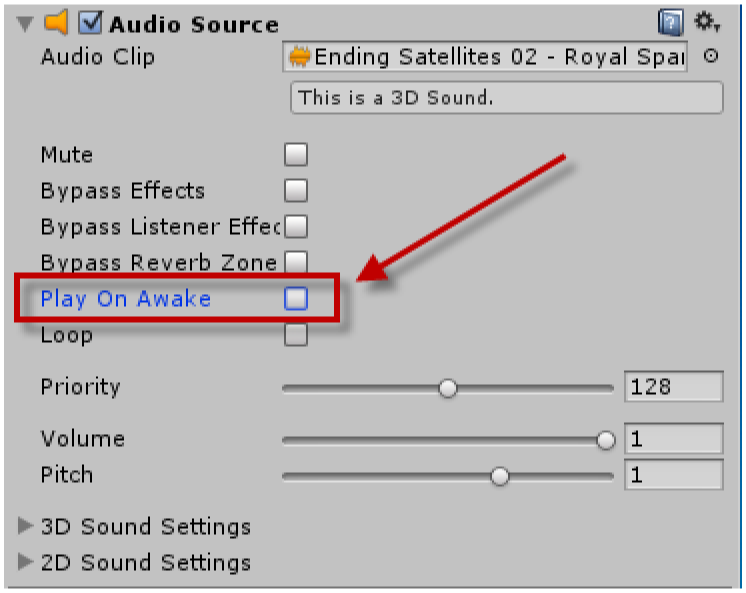
12.1. We need to assign the HelloWorldKoreography asset into the SimpleMusicPlayer component. This can be done by dragging our HelloWorldKoreography asset into the empty form field or by selecting the Unity load button on the right side of the empty form field, select the Assets tab rather than the scene tab, and then select the asset.



12.2. We have an *AudioSource*, but no associated *AudioClip*. To assign an *AudioClip* to the attached *AudioSource* either drag the *AudioClip* to the *AudioSource* form field or select the *Unity* load icon (the small circle to the right).



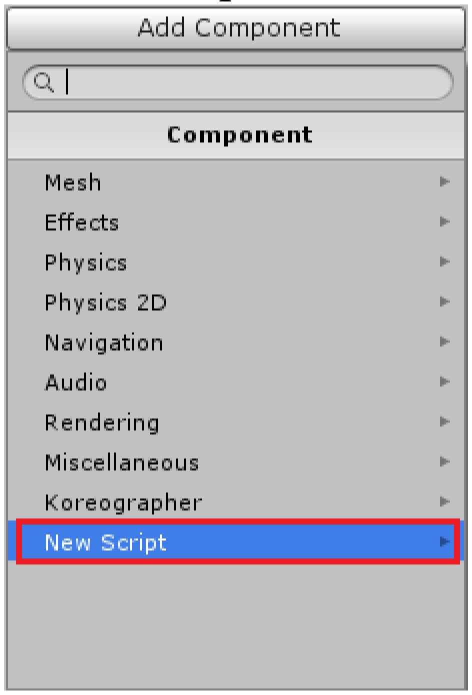
12.3. Our last step in setting up our *Koreographer* object in the scene is to set the *AudioSource* property ***Play On Awake*** *to unchecked.*



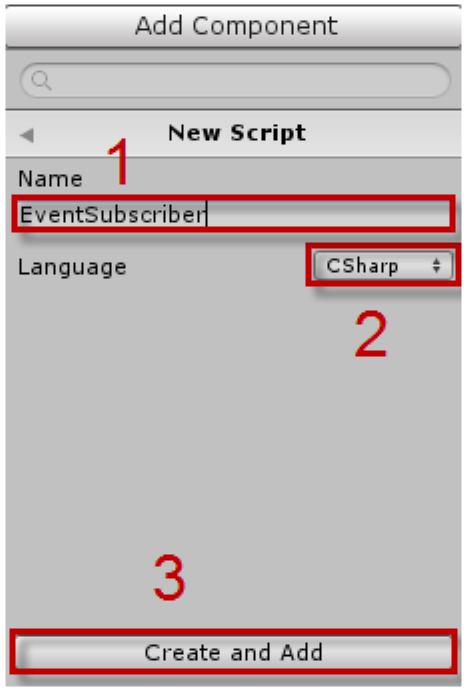
13. We now have a fully fledged *Koreographer* object ready to track and fire events. We just need a *GameObject* in the scene with a class setup to subscribe to the event.

13.1. Create another empty *GameObject* in the scene and name it **TestEventSubscriber**. This time we are going to add a C# script component to it that does not exist yet. To do this:

- Select **Add Component** under the *TestEventSubscriber* properties in the Unity Inspector and select the **New Script** menu item.

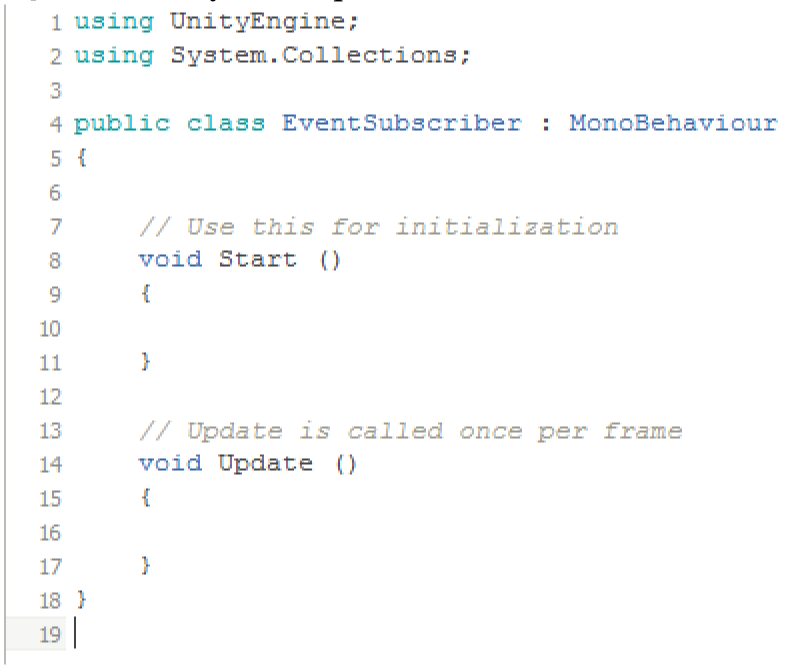


- Name the new script **EventSubscriber** and *switch the language to C#*. Confirm these settings and create/add the script to the *TestEventSubscriber* object by selecting **Create and Add**.



14. You will see the *EventSubscriber* script now attached to the GameObject as well as in the Project files. Double-click it in either location as our last task will be to modify the script to subscribe to our Koreography's events. The script will open in your preferred script editor (this may be MonoDevelop or Visual Studio).

14.1. 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 ()
    {

    }
}

```

14.2. Subscribing to *Koreographer* for events is as simple as calling in your code singleton instance (fancy name for a global object) of *Koreographer*.

14.3. There are three things we want to consider when subscribing to events:

- The *EventTrackID* we want this particular subscription to receive events from. This *EventTrackID* 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 function delegate method we wish to call. A function delegate defines the function much in the same way a type would. Think of it as the signature of a function. We will create a function that uses the same structure or signature as the defined delegates and when we register the *EventTrackID* with *Koreographer*, we will also register our custom correlating function.

Koreographer defines two function delegates we can use when creating functions that will be subscribed for events.

```
void KoreographyEventCallback(KoreographyEvent koreoEvent);
void KoreographyEventCallbackWithTime(KoreographyEvent koreoEvent, int sampleTime, int sampleDelta);
```

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* and two *int*s (sampleTime and sampleDelta). This is used for when you wish to receive the event and some information about where in the track the event occurred. *SampleTime* will be the time the sample was taken when the *KoreographyEvent* occurred while the sample delta will be the total number of samples since the last frame (similar to how *Time.deltaTime* provides the canonical time since the previous frame).

In this case, we will be creating a single function that matches the first delegate (only a single parameter of *KoreographyEvent*) and our statement will print a Debug statement to the console stating that it received the event. As you can see below, the function *FireEventDebugLog* matches the function delegate **KoreographyEventCallback.**

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

14.4. Our last task before we test will be to register the EventTrackID and the 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);
}
```

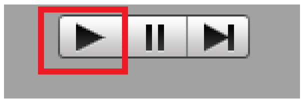
14.5. The completed version of the *EventSubscriber* class will look like this. Save the file and close out your script editor.

```
using UnityEngine;
using System.Collections;

public class EventSubscriber : MonoBehaviour
{

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

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

    }

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

```

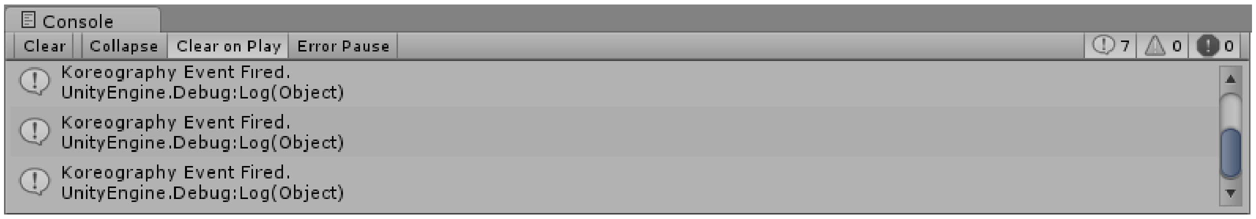
Running and Testing the Koreographer Functionality

15. Now that we are back in Unity, we can test to make sure our events are being received.

15.1. Press the Play button.



15.2. As the music plays and events are being fired you should see log statements made in the console from the *TestEventSubscriber*, like so.



```
Koreography Event Fired.
UnityEngine.Debug:Log(Object)
Koreography Event Fired.
UnityEngine.Debug:Log(Object)
Koreography Event Fired.
UnityEngine.Debug:Log(Object)
```