Record And Play Setup

This guide is meant to get you up and running as quick as possible with using Record And Play.

Many links in this document will point you to documentation located at https://recolude.gitlab.io/RecordAndPlay3D/. This is a website built from the code comments provided to you in this package. The documentation found there can be found throughout the source code inside this package.

There exists a small youtube series on getting you started with using the extension that can be found here.

For any issues please email eli.davis1995@gmail.com with the title of the email in the format of "Record and Play: [your bug]" for my filter to pick it up.

Recording Quick-Start

This guide corresponds to the video tutorial located here.

A recording consists of metadata, custom events, and subjects. The easiest way to build a recording is to make a recorder to do it for you. There are two different ways to create a recorder. The first way is through the editor by right clicking anywhere in your project and selecting **Create > Record and Play > Recorder** as seen in the Recorder documentation.

The other way to create a recorder is by using code like so:

```
var myRecorder = ScriptableObject.CreateInstance<Recorder>();
```

Once you have your recorder, you can register the subjects you want to be recorded. To do this you need to attatch a SubjectBehavior script to what ever GameObject you want to keep up with. There are helper functions for setting up this script called Build().

```
Recorder recorder = ScriptableObject.CreateInstance<Recorder>();
GameObject objectToTrack = GameObject.CreatePrimitive(PrimitiveType.Cube);
SubjectBehavior.Build(objectToTrack, recorder);
```

Calling Build attaches the script to what ever GameObject you passed in and registers it with the recorder passed in. For details on what framerate and minimumDelta is, please refer to SubjectRecorder. The alternative way to create a SubjectBehavior is to manually attach an instance of it to whatever GameObject you want to track and drag in a recorder for it to use in the inspector. To actually begin a recording, you must start the recorder by calling Start(). While the recorder is capturing events, you can log recording custom events or subject custom events. Then to finally create a recording you need to call Finish(), which will then pop you out a Recording.

```
Recorder recorder = ScriptableObject.CreateInstance<Recorder>();
GameObject objectToTrack = GameObject.CreatePrimitive(PrimitiveType.Cube);
SubjectBehavior subject = SubjectBehavior.Build(objectToTrack, recorder);
recorder.Start()

// ...stuff happens....
subject.CaptureCustomEvent("custom subject event", "event details");
recorder.CaptureCustomEvent("custom recorder event", "other event details");

// Finish up the recording.
Recording myRecording = recorder.Finish();

// Now you can do something with the recoding such as save it or play it back
myRecording.SaveToAssets("Some name for your asset")
PlaybackBehavior.Build(myRecording).Play();
```

Example

```
using UnityEngine;

// Import the namespace that contains the Recorder and SubjectBehavior Class.
using EliCDavis.RecordAndPlay.Record;

///<summary>
/// This script is meant to act as a demo on creating recordings. It creates 3
/// boxes and attaches subject behavior scripts to them. It then starts the
/// recorder. The user can choose to stop the recorder by clicking the save
/// button. Clicking the save button takes the recording and saves it to assets
/// folder in the project.
///</summary>
public class RecordingExample : MonoBehaviour
{

    // The recorder in charge of keeping up with all the subjects in the
    //scene.
    Recorder recorder;

    // Start is called before the first frame update
    void Start()
    {
        // Because the recorder is a scriptable object, we have to create
        // it using CreateInstance().
        recorder = ScriptableObject.CreateInstance<Recorder>();

        int numberOfSubjects = 3;
        for (int i = 0; i < numberOfSubjects; i++)
        {
            var cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
            cube.AddComponent<Rigidbody>();
            cube.transform.position = new Vector3((i * 2) - 1.5f, 3, 0);
```

```
            // Create a SubjectBehavior instance and attatch it to the
            // cube. Doing this automatically registers the subject
            // with the recorder passed in.
            SubjectBehavior.Build(cube, recorder);
        }

        // Start the recorder so it will capture all events occuring both
        // to it and the different subjects registered.
        recorder.Start();
    }

    private void OnGUI()
    {
        // If the recorder is currently recording and the player clicks
        // save...
        if (recorder.CurrentlyRecording() && GUILayout.Button("Save"))
        {
            // Create a recording with all captured events up to this
            // point and stop the recorder from accepting any new
            // event captures.
            var recording = recorder.Finish();

            // Take the recording we just created and save it to the
            // assets folder inside our project.
            recording.SaveToAssets("Recording Example");
        }
    }
}
```

Playback Quick-Start

This guide corresponds to the video tutorial located here.

The first step to playback is first having a recording to actually playback. Once you have reference to a recording you can use it to build a PlaybackBehavior instance using it's Build() function.

```
PlaybackBehavior myPlaybackBehavior = PlaybackBehavior.Build(myRecording);
```

With a reference to PlaybackBehavior you can call Play, Pause, or Stop to control the recording. You can also change the speed the recording is being played back with SetPlaybackSpeed. We can set the time we are currently at in the playback of the recording using SetTimeThroughPlayback. By default the playback will instantiate cubes as actors to represent the subjects inside the recording. You can set what is used to represent a subject by creating a class to implement the IActorBuilder interface.

```
using System.Collections;
using UnityEngine;
```

```
// Import the playback namespace for access to PlaybackBehavior
using EliCDavis.RecordAndPlay.Playback;

public class ActorBuilderExample : IActorBuilder
{
    // Creates an actor that uses a sphere to represent itself.
    public Actor Build(int subjectId, string subjectName, Dictionary<string,
string> metadata)
    {
        return new Actor(GameObject.CreatePrimitive(PrimitiveType.Sphere));
    }
}
```

You can also choose to add custom functions to be called when a custom event occurs by implementing the IPlaybackCustomEventHandler interface. If the subject passed in is null, then it is a global custom event.

```
using System.Collections;
using UnityEngine;

// Import the playback namespace for access to PlaybackBehavior
using EliCDavis.RecordAndPlay.Playback;

public class CustomEventHandlerExample : IPlaybackCustomEventHandler
{
    // This method satisfies the IPlaybackCustomEventHandler interface and
    // debugs events.
    public void OnCustomEvent(SubjectRecording subject, CustomEventCapture
customEvent)
    {
        if (subject == null)
        {
            Debug.LogFormat("Global Custom Event: {0} - {1}", customEvent.Name,
customEvent.Contents);
        }
        else
        {
            Debug.LogFormat("Custom Event For {0}: {1} - {2}",
subject.SubjectName, customEvent.Name, customEvent.Contents);
        }
    }

}
```

You can pass in these implemented interfaces when building the playback behavior like so:

```
PlaybackBehavior.Build(recording, new ActorBuilderExample(), new
CustomEventHandlerExample(), true);
```

If you want to control where in the scene the playback is occuring, you can set the position of the transform that the PlaybackBehavior is attatched to. This applies to rotation and scale as well.

Example

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// Import the record and play namespace for access to the Recording class.
using EliCDavis.RecordAndPlay;

// Import the playback namespace for access to PlaybackBehavior
using EliCDavis.RecordAndPlay.Playback;

/// <summary>
/// This script is meant for providing playback controls through a gui to any
/// recording passed in. This class also logs custom events as they happen
/// through the playback. This class demonstrates how you can implement the
/// IActorBuilder and IPlaybackCustomEventHandler for building custom playback.
/// </summary>
public class PlaybackExample : MonoBehaviour, IActorBuilder,
IPlaybackCustomEventHandler
{

    // The recording we are going to be playing back. Marked SerializeField
    // so it ican be assigned in the editor.
    [SerializeField]
    Recording recording;

    // A class scoped reference to PlaybackBehavior that is in control of
    // animating the playback of the recording.
    PlaybackBehavior playbackBehavior;

    void Start()
    {
        // Create an instance of the playback behavior which will be
        // responsible for animating the recording we pass in. We also
        // tell it to use this class for both creating the actors that
        // will represent the subjects recorded, and to use this class
        // for responding to custom events in the recording.
        playbackBehavior = PlaybackBehavior.Build(recording, this, this, true);
    }

    // This method satisfies the IActorBuilder interface. The function takes
    // the unique id for the subject, the name of the subject, and any
    // assigned metadata. With that it should return a reference to a GameObject
that
    // exists in the current scene to act as a representation for the
    // subject. There is an alternative to building the actor which
    // includes an optional argument that is the handler for custom events
    // that occurred to the subject (not shown here).
```

```
    public Actor Build(int subjectId, string subjectName, Dictionary<string,
string> metadata)
    {
        return new Actor(GameObject.CreatePrimitive(PrimitiveType.Sphere));
    }

    // This method satisfies the IPlaybackCustomEventHandler interface. This
    // function will be called as the current time in the playback
    // progresses over the timestamp of the custom event. If the custom
    // event belongs to a specific subject then details of the subject will
    // be passed in. Else the subject will be null and all you will be left
    // with is the custom event itself.
    public void OnCustomEvent(SubjectRecording subject, CustomEventCapture
customEvent)
    {
        if (subject == null)
        {
            Debug.LogFormat("Global Custom Event: {0} - {1}", customEvent.Name,
customEvent.Contents);
        }
        else
        {
            Debug.LogFormat("Custom Event For {0}: {1} - {2}",
subject.SubjectName, customEvent.Name, customEvent.Contents);
        }
    }

    private void OnGUI()
    {
        if (playbackBehavior.CurrentlyStopped())
        {
            if (GUILayout.Button("Play"))
            {
                playbackBehavior.Play();
            }
        }
        else if (playbackBehavior.CurrentlyPlaying())
        {
            if (GUILayout.Button("Pause"))
            {
                playbackBehavior.Pause();
            }
            if (GUILayout.Button("Stop"))
            {
                playbackBehavior.Stop();
            }
        }
        else if (playbackBehavior.CurrentlyPaused())
        {
            if (GUILayout.Button("Play"))
            {
                playbackBehavior.Play();
            }
            if (GUILayout.Button("Stop"))
```

```
            {
                playbackBehavior.Stop();
            }
        }
    }
}
```