DTT
APPS WEB GAMES

SINCE 2010

Audio Recording

Documentation | 24-06-22

Table of Contents

1. Get started quickly 3

2. Introduction 4

3. Set-Up 5

4. Editor 7

5. API 8

    RecordingsManager 8

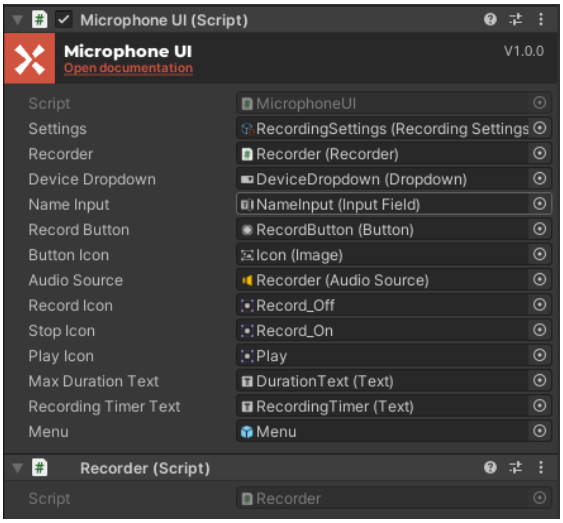
    WAVSaver 9

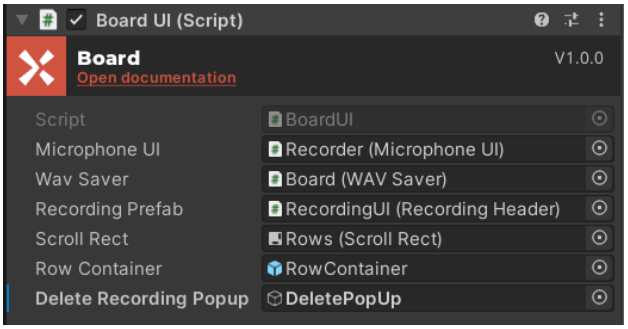
6. Known Limitations 10

1. Get started quickly

This is a summary to help you get started quickly. More details are provided below.

1. Open a new scene in **Unity** and add a **Canvas** object.
2. Open the prefab folder located at *'Demo/Prefabs/'* and add the Board prefab to the **Canvas**. All references should be correctlyset by default.
    a. In case a reference is missing, here is how the components are supposed to look:

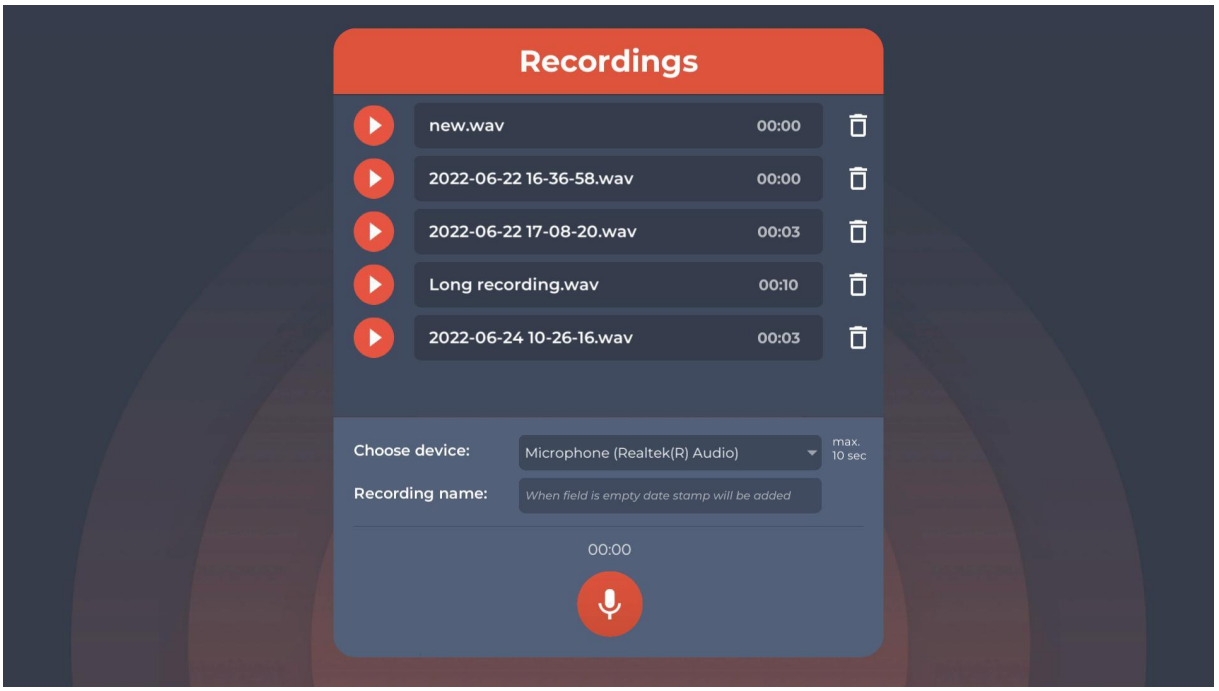




3. Configure the RecordingSettings **ScriptableObject** to fit your needs. You can create a new one by right-clicking on the project and selecting *'Create/DTT/AudioRecording/RecordingSettings'.*
4. Then pass it to the MicrophoneUI component. You can learn more about the settings at 4. Editor.

2. Introduction

DTT Audio Recording is a **Unity** asset, which aims to provide you with the foundation needed to record and manage audio clips from your selected device's microphone. The asset includes a demo scene with UI classes, which gives you an example of how you can use the framework in your project. The logic to save, load, display, delete, play, and record new audio recordings is included. It includes a demo scene that can be used as an example implementation. The code is extendable, so you can use it for your own custom recording classes and saving format.



3. Set-Up

This section will guide you through the process of implementing a custom Audio Recording system.

1. Open a scene in **Unity**. Create a **GameObject** with a script for managing the UI for the Recordings.
2. Create a new data class that derives from the Recording base class to store the different properties you would like each recording to have. Screenshot from the AudioClipRecording class used in the demo:

```
using UnityEngine;

namespace DTT.AudioRecording
{
    /// <summary>
    /// Data class for Recordings with an AudioClip.
    /// </summary>
    public class AudioClipRecording : Recording
    {
        /// <summary>
        /// AudioClip property.
        /// </summary>
        public AudioClip Clip { get; private set; }

        /// <summary>
        /// Constructor with a custom name.
        /// </summary>
        /// <param name="name">AudioClip name.</param>
        /// <param name="duration">AudioClip duration.</param>
        /// <param name="clip">The AudioClip itself.</param>
        public AudioClipRecording(string name, float duration, AudioClip clip) : base(name, duration) => Clip = clip;

        /// <summary>
        /// Constructor with a timestamp as a name.
        /// </summary>
        /// <param name="duration">AudioClip duration.</param>
        /// <param name="clip">The AudioClip itself.</param>
        public AudioClipRecording(float duration, AudioClip clip) : base(duration) => Clip = clip;
    }
}
```

3. Create or add your own API for saving and loading. The class should implement the IRecordingSaver<T> interface, where T is your Recording class. Screenshot from the WAVSaver class used in the demo:

```
namespace DTT.AudioRecording
{
    /// <summary>
    /// Recording saver for the WAV format.
    /// </summary>
    public class WAVSaver : MonoBehaviour, IRecordingSaver<AudioClipRecording>
    {
        /// <summary>
        /// Collection with all AudioClipRecordings.
        /// </summary>
        public List<AudioClipRecording> Recordings { get; private set; }

        /// <summary>
        /// Deletes a recording and invokes a callback event.
        /// </summary>
        /// <param name="path">File path of the recording.</param>
        /// <param name="recording">Recording to delete.</param>
        /// <param name="callback">Callback event to invoke.</param>
        public void Delete(string path, AudioClipRecording recording, Action callback)...

        /// <summary>
        /// Saves a new recording and invokes a callback event with the saved Recording.
        /// </summary>
        /// <param name="path">File path for saving.</param>
        /// <param name="recording">Recording to save.</param>
        /// <param name="callback">Callback event to invoke.</param>
        public void Save(string path, AudioClipRecording recording, Action<AudioClipRecording> callback)...

        /// <summary>
        /// Loads the data from a given path and invokes a callback event with an array of recordings.
        /// </summary>
        /// <param name="path">Folder path.</param>
        /// <param name="callback">Callback event.</param>
        public void Load(string path, Action<AudioClipRecording[]> callback)...
```

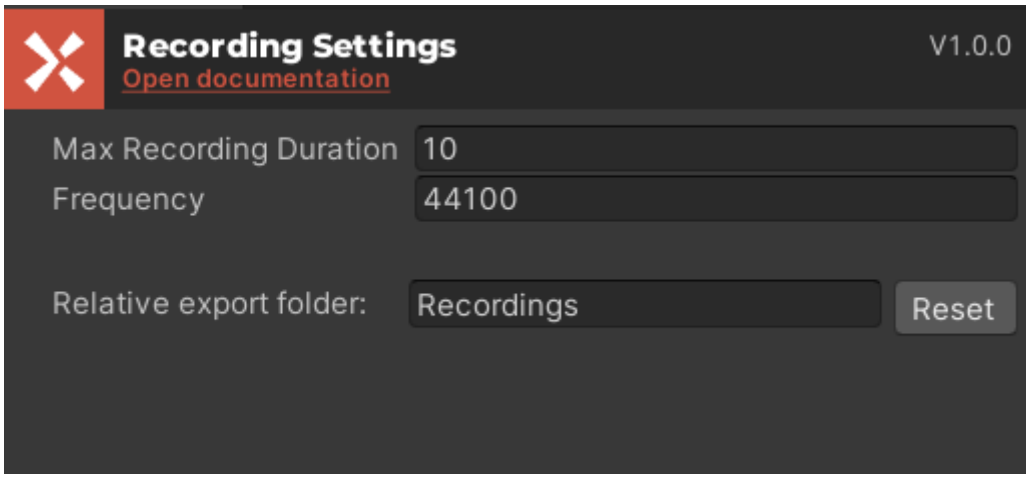
4. Then in your UI manager class add a new **RecordingsManager<T>**, where **T** is your **Recording** class. Initialize the **RecordingManager** by passing the **IRecordingSaver** class you created to the constructor. Screenshot from the **BoardUI** class used in the demo.

```
/// <summary>
/// Recording manager class for AudioClipRecordings.
/// </summary>
private RecordingsManager<AudioClipRecording> _recordingsManager;
```

```
/// <summary>
/// Initializes the necessary objects.
/// </summary>
private void Awake()
{
    _recordingsManager = new RecordingsManager<AudioClipRecording>(_wavSaver);
```

5. You are now ready to use all of the APIs described in 5. API. Check the code in **BoardUI** and **MicrophoneUI** for an example implementation,

4. Editor



The RecordingSettings **ScriptableObject** is used to configure the MicrophoneUI class.

1. Max Recording Duration:

   The maximum length of the new Recording's **AudioClip**.

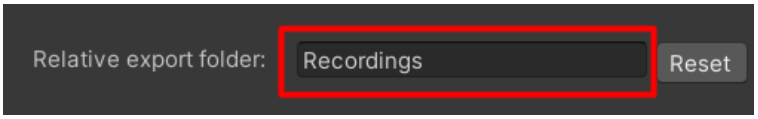
2. Frequency:

   This setting determines the sample rate of the **AudioClip** produced by the recording.

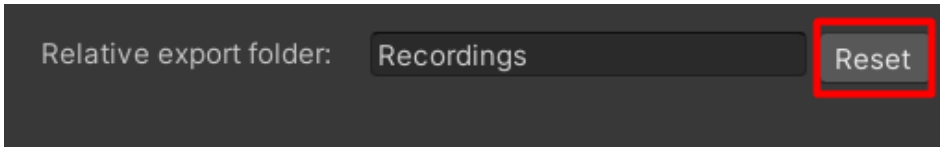
3. Export folder:

   This is the folder from where all Recordings are loaded and new Recordings are saved.

   a. Use the text field to write the name of the folder, where you would like to load and save recordings.



   b. The "Reset" button resets the folder path to the default one (Application.persistentDataPath/Recordings).



5. API

RecordingsManager

Manager class for the recordings.

Property Name	Type	Description
RecordingsLoaded	event	Event invoked when the recordings have been loaded.
RecordingSaved	event	Event invoked when the recording has been saved.
RecordingDeleted	event	Event invoked when the recording has been deleted.

Method name	Return Type	Parameters	Description
LoadRecordings	void	string path	Used to load all recordings by calling IRecordingSaver.Load.
SaveRecording	void	string path, T recording	Used to save a recording by calling IRecordingSaver.Save.
DeleteRecording	void	string path, T recording	Used to delete a recording by calling IRecordingSaver.Delete.

WAVSaver

Custom recording saver that uses the WAV format and implements the **IRecordingSaver** *interface.*

Property Name	Type	Description
Recordings	List<AudioClipRecording>	Collection with all AudioClipRecordings that are loaded.

Method name	Return Type	Parameters	Description
Load	void	string path, Action<AudioClipRecording[]> callback	Used to load all recordings by calling IRecordingSaver.Load. Invokes the callback event with all loaded recordings.
Save	void	string path, AudioClipRecordign recording, Action<AudioClipRecording> callback	Saves a new recording and invokes the callback event with the saved recording.
Delete	void	string path, AudioClipRecording recording, Action callback	Deletes a recording and invokes the callback event.

6. Known Limitations

- Does not inherently support remote APIs.

7. Support and feedback

If you have any questions regarding the use of this asset, we are happy to help you out. Always feel free to contact us at:

unity-support@d-tt.nl

(We typically respond within 1-2 business days)

We are actively developing this asset, with many future updates and extensions already planned. We are eager to include feedback from our users in future updates, be they 'quality of life' improvements, new features, bug fixes or anything else that can help you improve your experience with this asset. You can reach us at the email above.

Reviews and ratings are very much appreciated as they help us raise awareness and to improve our assets.

DTT stands for Doing Things Together

DTT is an app, web and game development agency based in the centre of Amsterdam. Established in 2010, DTT has over a decade of experience in mobile, game, and web based technology.

Our game department primarily works in Unity where we put significant emphasis on the development of internal packages, allowing us to efficiently reuse code between projects. To support the Unity community, we are publishing a selection of our internal packages on the Asset Store, including this one.

More information about DTT (including our clients, projects and vacancies) can be found here:

https://www.d-tt.nl/en/