Table of contents:

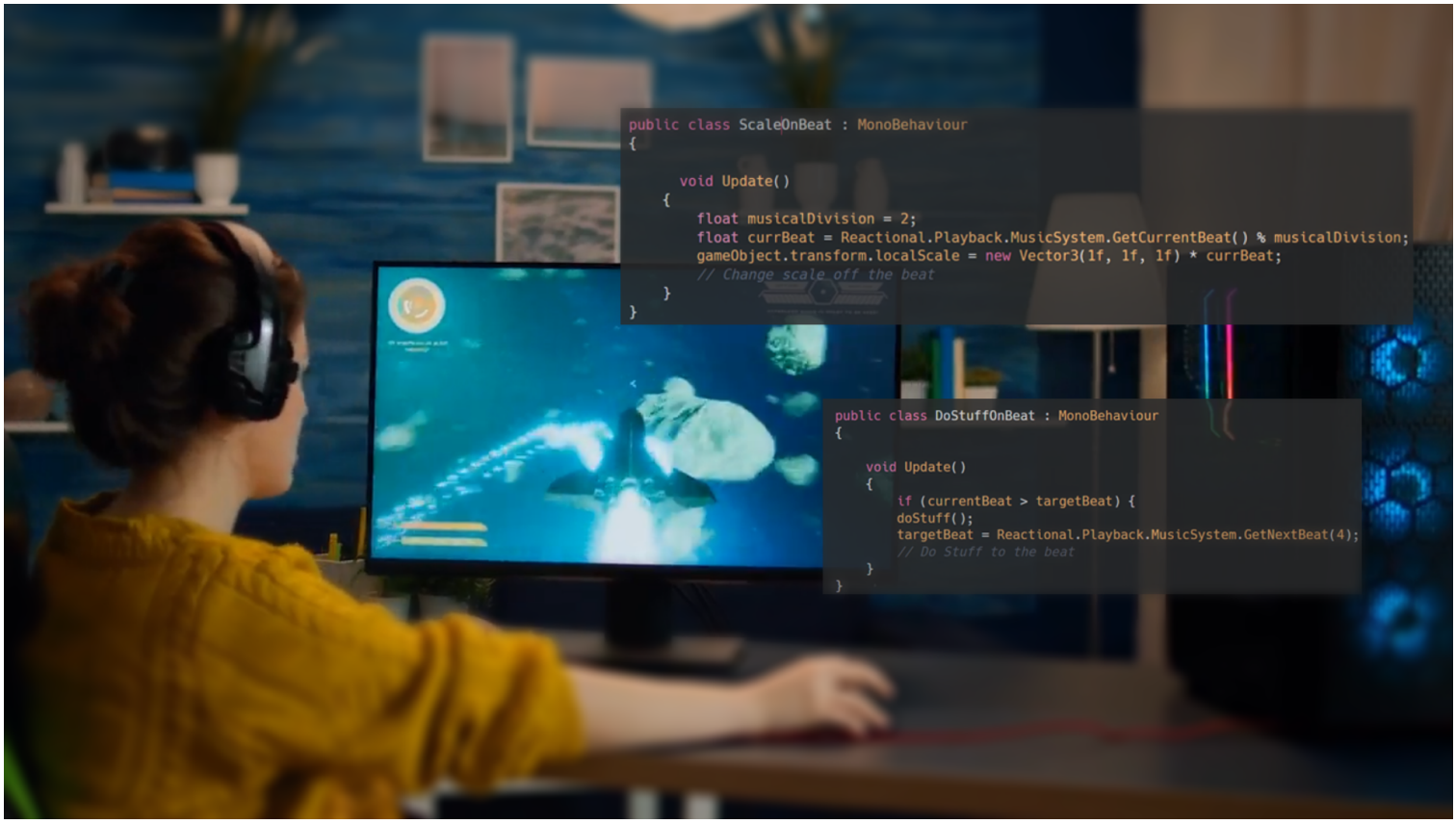
- Unity
  - Download Games and Demo Scene
  - Bespoke Reactional Themes for your game project
- Unity Quick Start Guide
  - Step 1 Download the plugin.
  - Step 2 Install in Unity.
  - Step 3 Click Import.
  - Step 4 Unzip and drag downloaded Reactional Project Bundle folder into Unity
  - Step 5 Add Reactional Manager to the scene
  - Step 6 Inspect the Reactional Manager GameObject
- Best Practices
  - Summary
  - Version Support
  - Considerations
    - Simple Logic, A Player Prefab or just a Few Reactional dependent Objects in the Scene
    - Big projects, lots of Objects and complex logic
  - Other Tips
  - Build Support
  - Reactional Integration
  - Benchmark
- Known Issues and Troubleshooting
- Report Issue
    - On Win
    - On Mac
    - Android
    - iOS
- Adding the Reactional SDK
  - Step 1. Download unitypackage
  - Step 2. Installation
  - Step 3. You're set!
- Unity Changelog
  - Plugin Version 1.0rc11 - 24-07-01
- Overview
    - Setup
    - Playback.Theme

    - Playback.Playlist
    - Playback.MusicSystem
- Setup
    - Properties
        - Samplerate
        - BlockSize
        - Lookahead
        - IsValid
        - AllowPlay
    - Methods
        - InitAudio
        - UpdateBundles
        - LoadBundles
        - LoadBundle
        - LoadSection
        - LoadPlaylist
        - LoadTrack
        - LoadTheme
    - Enumerators
        - LoadType
        - UpdateMode
        - AudioOutputMode
- Playback.Theme
    - Methods
        - GetThemeID
        - IsLoaded
        - Play
        - Reset
        - Stop
        - State
        - GetControls
        - CurrentBeat
        - TempoBpm
        - SetControl
        - GetControl
        - GetNumParts
        - GetCurrentPart

        - GetPartName
        - GetPartOffset
        - GetPartDuration
        - GetNumBars
        - GetCurrentBar
        - GetBarOffset
        - GetBarDuration
        - TriggerStinger
        - InstrumentOverride
        - GetOverridableInstruments
    - Properties
        - Volume
- Playback.Playlist
    - Methods
        - CurrentBeat
        - TempoBpm
        - GetTrackID
        - GetCurrentTrackMetadata
        - GetCurrentTrackInfo
        - GetSelectedTrackIndex
        - IsLoaded
        - PlayTrack
        - PlayTrackByID
        - Next
        - Prev
        - Random
        - GetState
        - Stop
        - Play
        - FadeOut
        - FadeIn
        - GetNumParts
        - GetCurrentPart
        - GetPartName
        - GetPartOffset
        - GetPartDuration
        - GetNumBars

        - GetCurrentBar
        - GetBarOffset
        - GetBarDuration
    - Properties
        - Volume
        - IsPlaying
        - State
- MusicSystem
    - Methods
        - ScheduleAudio
        - GetEngine
        - GetTimeToBeat
        - GetRealTimeToBeat
        - GetNextBeat
        - GetNextBeatAbsolute
        - GetCurrentBeat
        - GetCurrentMicroBeat
        - GetTempoBpm
        - BeatsToSeconds
        - DuckMusic
    - Properties
        - PlaybackAllowed
    - Enumerators
        - Quant
        - PlaybackState
    - Classes
        - WaitForNextBeat
- Engine
    - ReactionalEngine
        - RouteNoteOn
        - RouteNoteOff
        - RouteStingerEvent
        - RouteBarBeat
        - RouteAudioEnd

Unity



Reactional Music's plug-in for Unity enables any game developer or sound designer to create interactive music for their game.

With Reactional's generative music technology, you can create adaptive sound and music note by note in real time based on any in-game events. Reactional's Unity plug-in also allows you to set in-game controls for musical events and developer-curated music personalization for the gamer. It lets you control the theme, music, game mechanics, animations, effects, or UI of your choice.

Simply download the plug-in, choose your interactive music and musical theme on the 🎵 **Reactional Platform**, and start with interactive sound design in just minutes. Lets go make some games: Unity Quick Start Guide

Download Games and Demo Scene

For an instant insight and understanding of what you can achieve Reactional, check out our Unity Games Showcase and Demo Scene

DOWNLOAD GAMES

Download our Games and Demo Scenes here:

DemoScene **Solitude**

Unity Showcase Scene

Bespoke Reactional Themes for your game project

You can create and upload your own Reactional Music Themes, creating your own interactive music. Find out more about how to do this with Reactional Composer Or you can also:

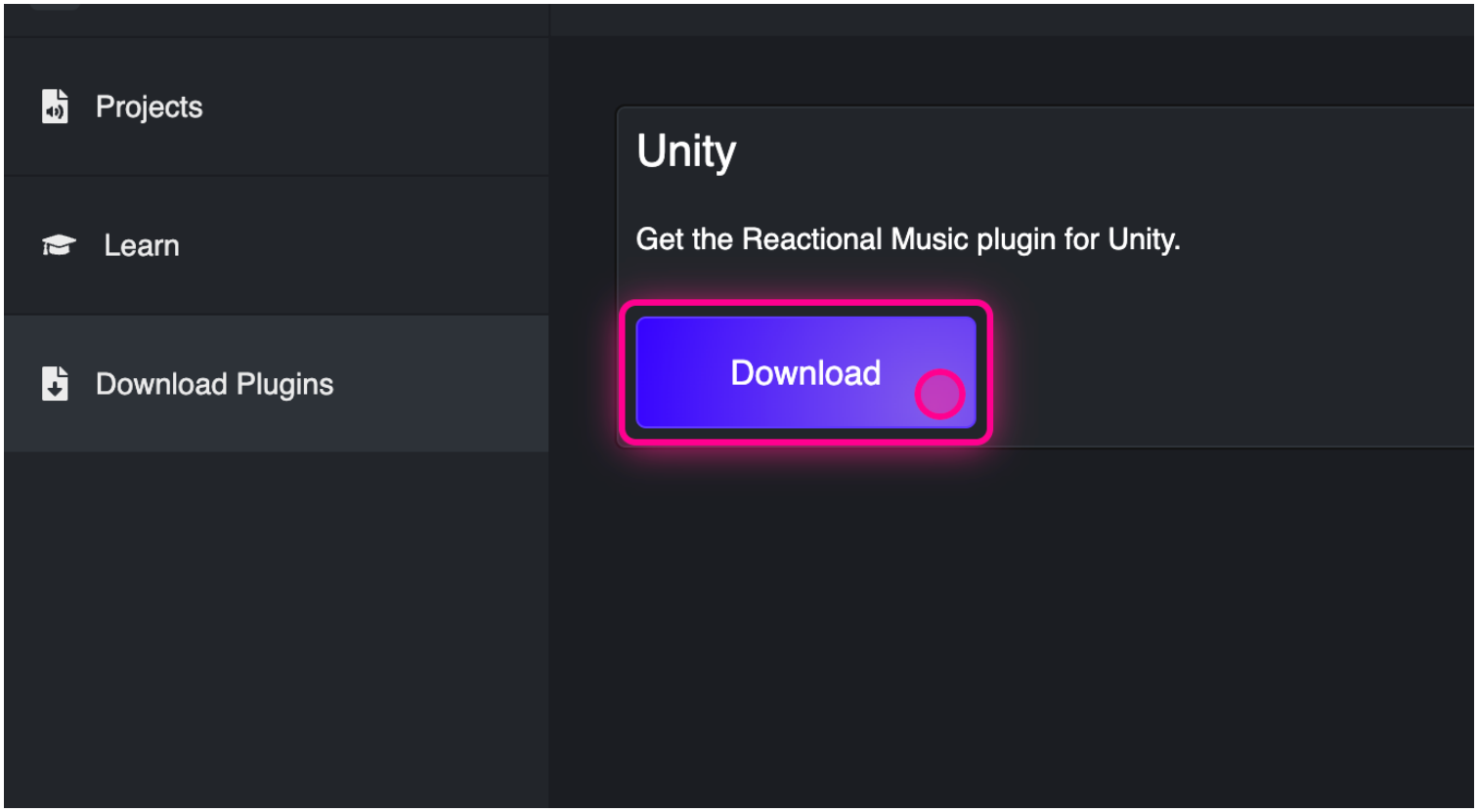
Join our Discord if you want to have your own bespoke Reactional Theme for your game.

Reactional Music Trailer | Game scene with Reactional Music

Unity Quick Start Guide

Step 1 Download the plugin.

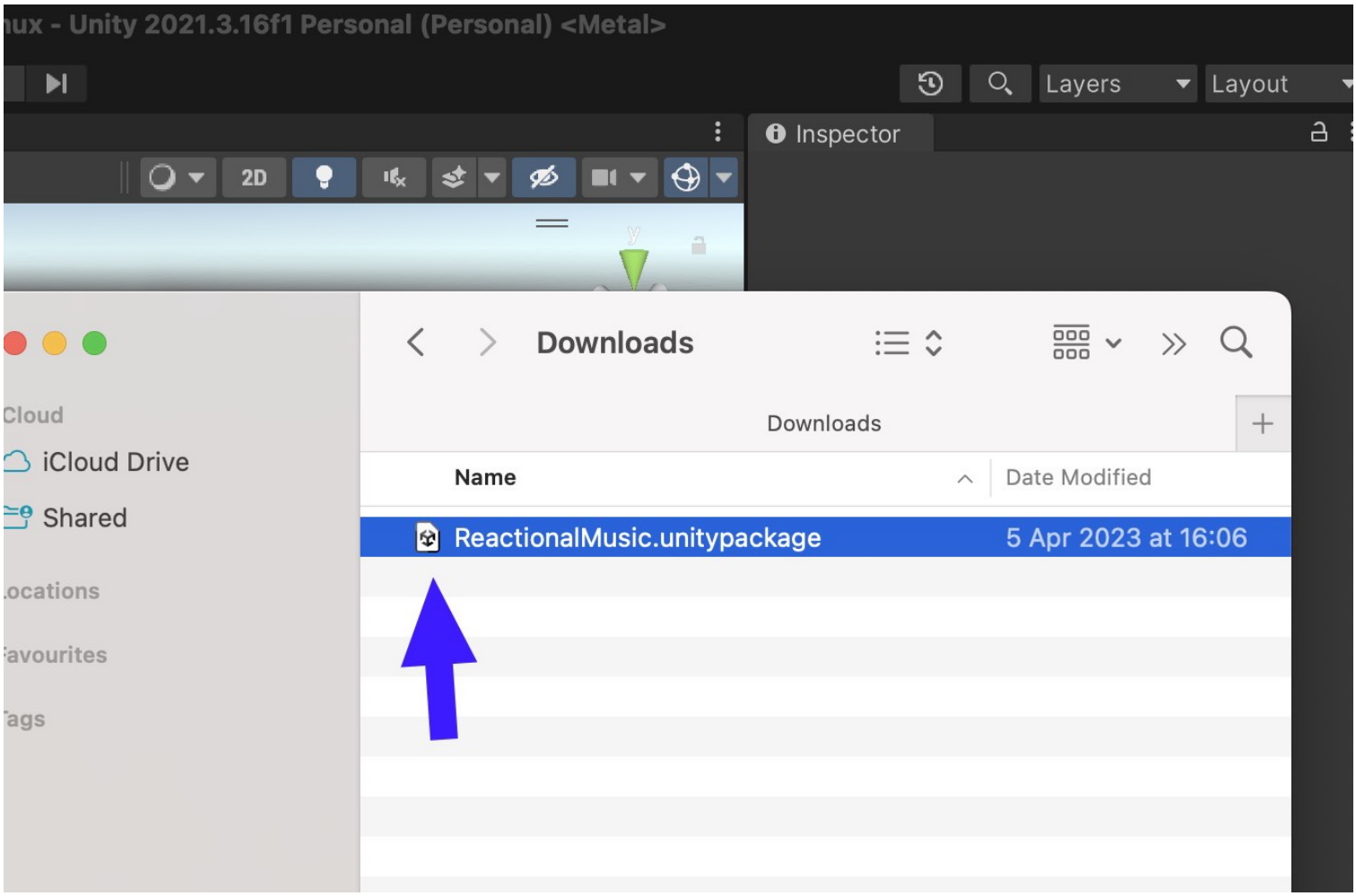
On the Reactional Platform you will find the Unity Plugin under Download Resources.



> **INFO**
>
> Sign up at the platform and Download the latest version of the Unity Plugin **here**

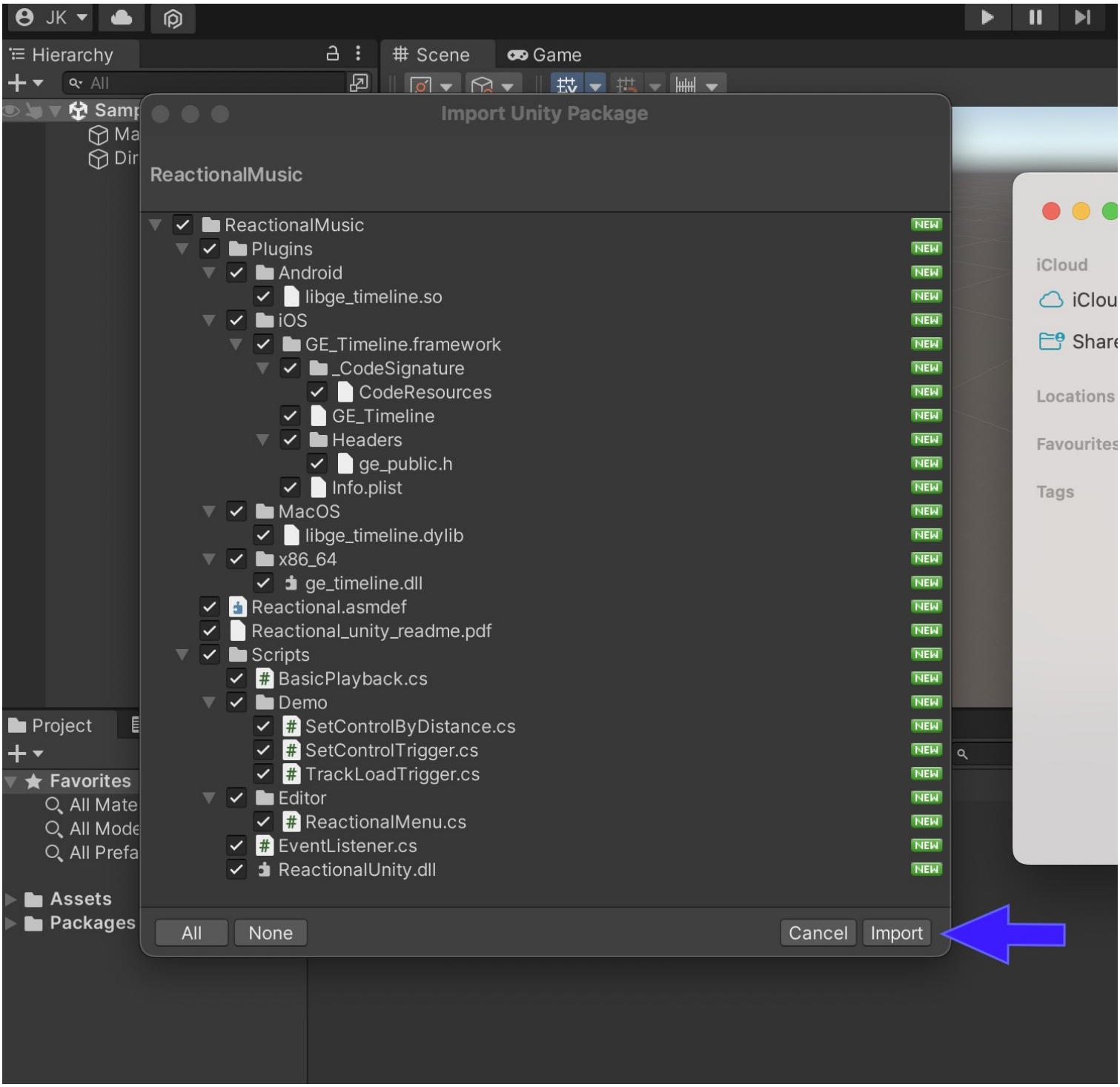
Step 2 Install in Unity.

With Unity Open, now open the folder in your download path and double-click the Reactional Music .unitypackage



Step 3 Click Import.

When in Unity, Reactional Music should popup with an option to "Import". Just click on import.



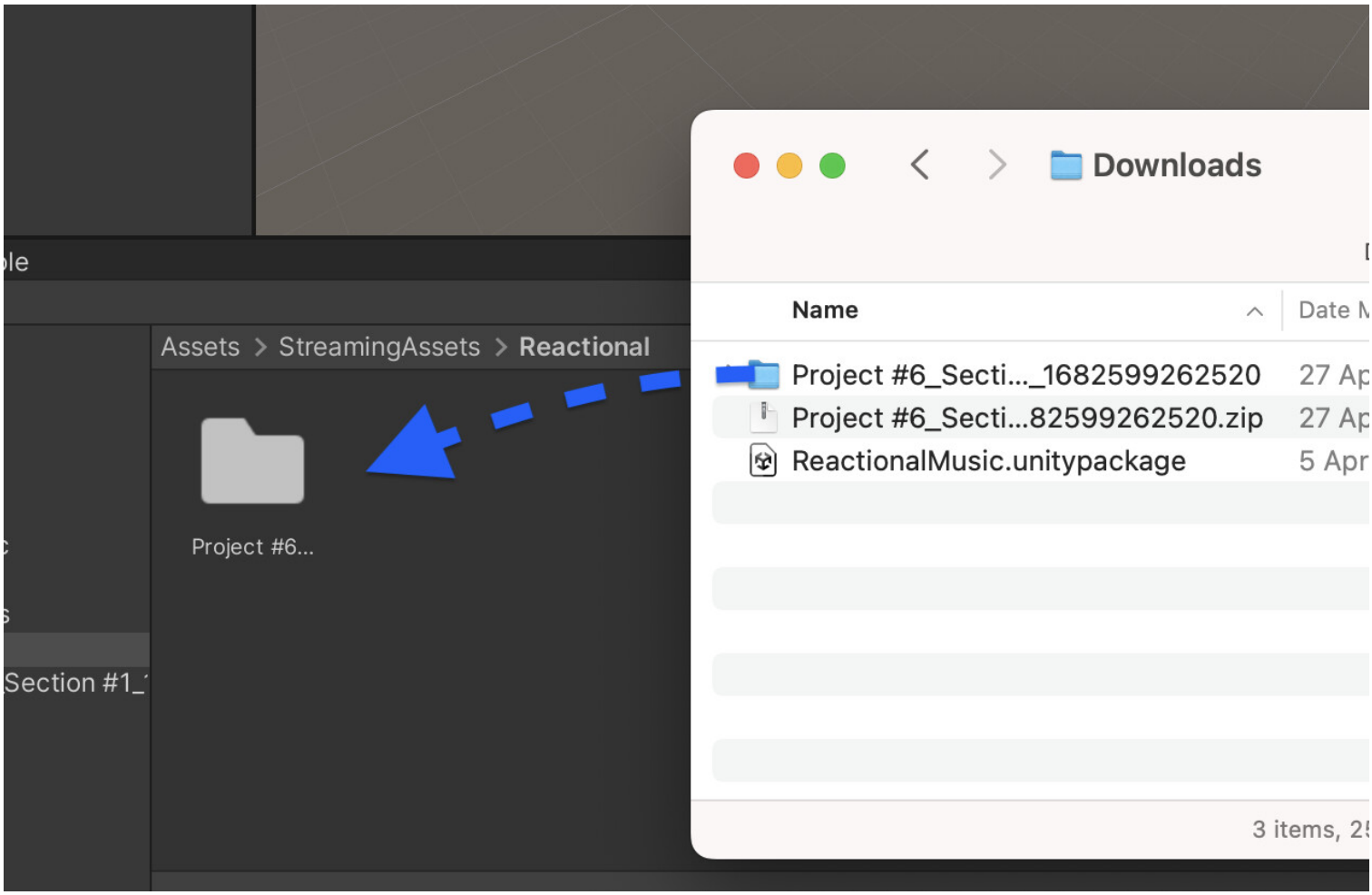
Step 4 Unzip and drag downloaded Reactional Project Bundle folder into Unity

Now lets find the bundle of your choice that you want to use. If you did not pick one yet head over back to the platform: **Reactional For Developers**

When you have your songs and theme ready and downloaded as a Zip file, extract those files into a folder. I chose to call my folder Project #6 as that was the name that i also used on the platform for this Reactional Bundle.

> **WARNING**
>
> When downloading test bundles they will expire after 30 days and will need to be re-downloaded.

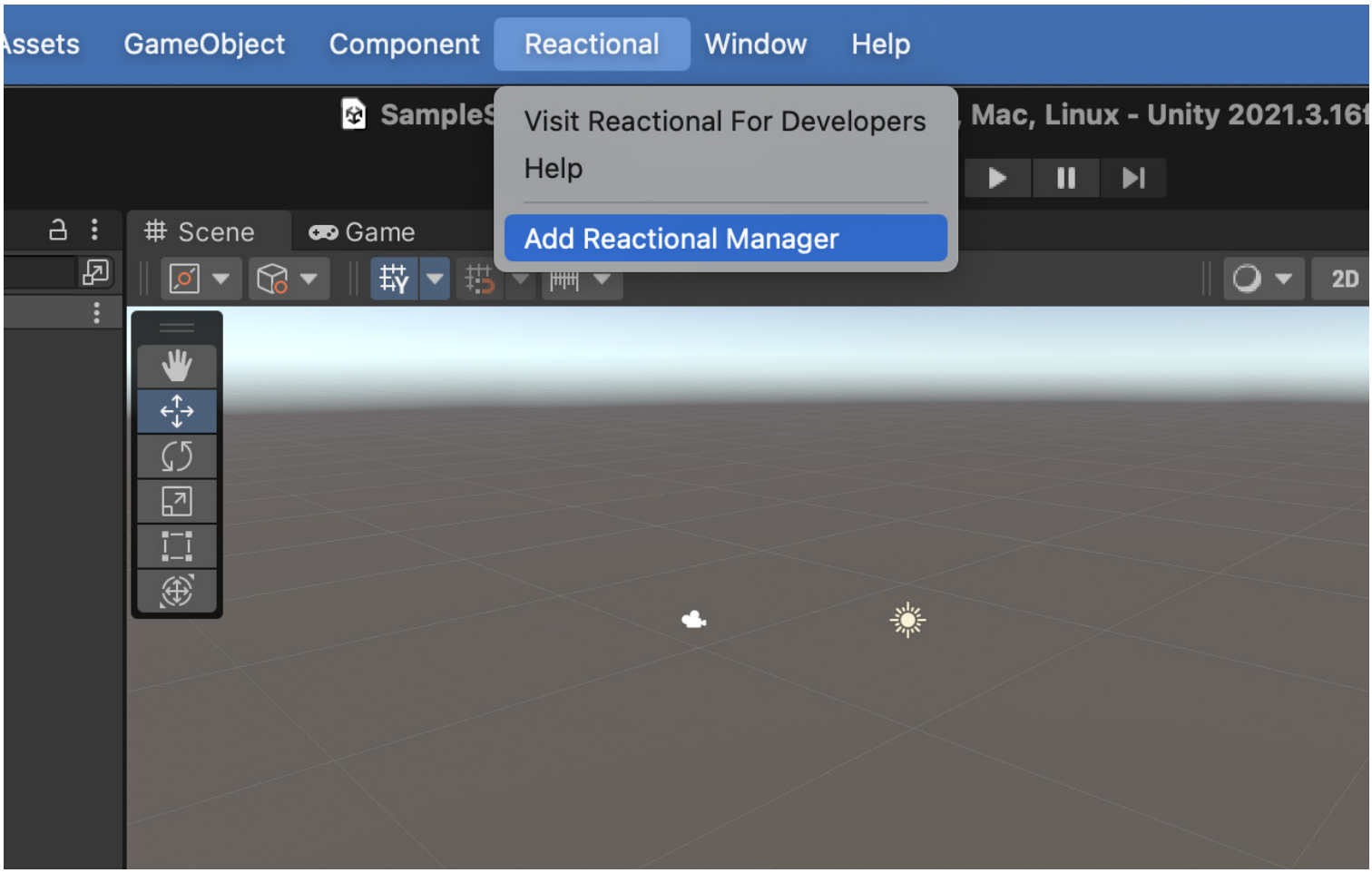
Simply Drag and Drop the files into the unity Asset>StreamingAsset>Reactional folder that was created for you when you installed the plugin. Make sure your bundle is dragged and droppped excatly here.



Step 5 Add Reactional Manager to the scene

To finish it all of and make it work. Add a Reactional Manager Object to your Unity Scene Hierarchy.

There is two ways to do this: Either via the pop-up dialog that will show automatically when opening scenes that do not yet have it. Or through the top menu on the left side of Window. Reactional --> Add Reactional Manager



Step 6 Inspect the Reactional Manager GameObject

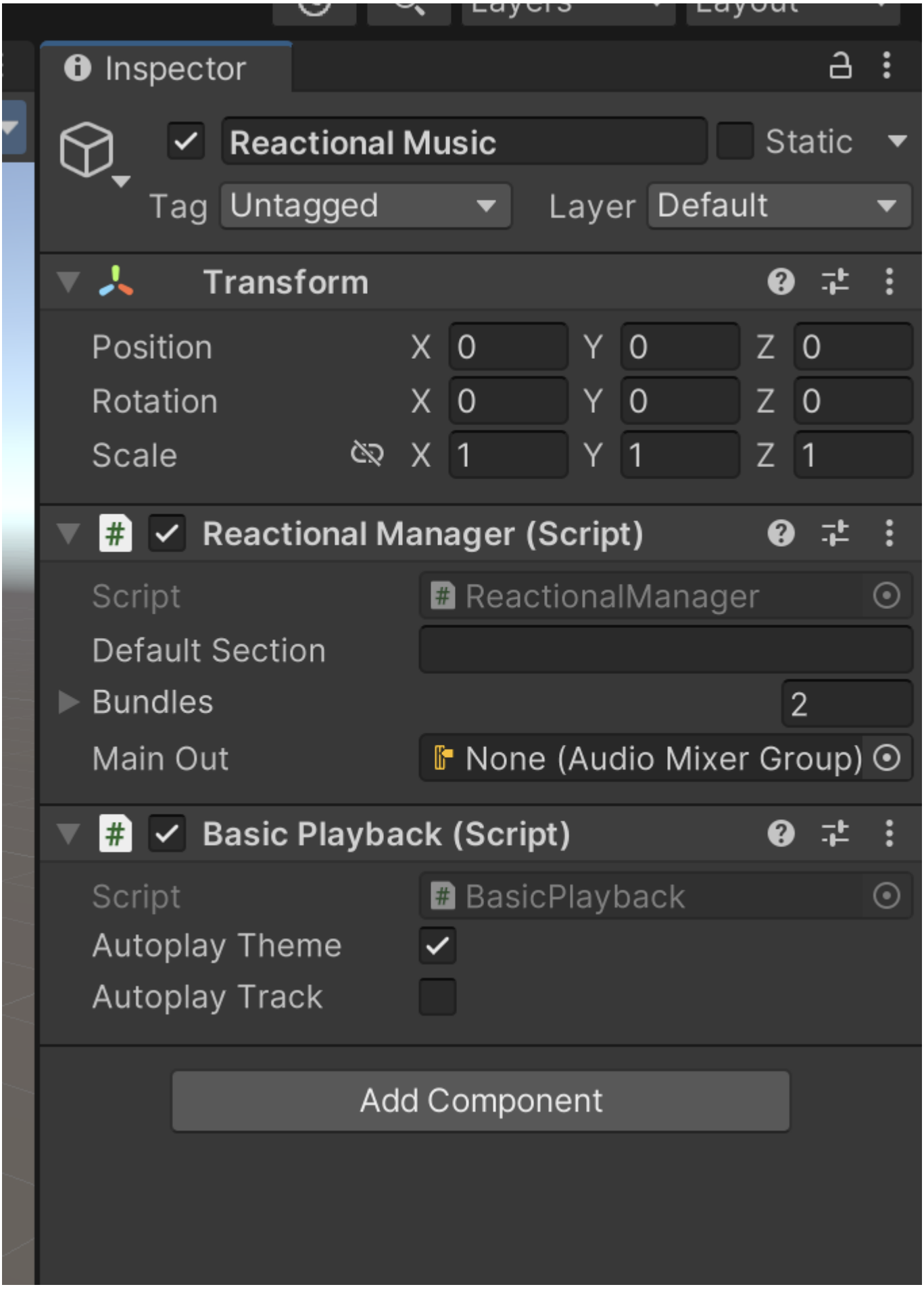
If no Default Section has been set, the manager will automatically pick the first defined, in the downloaded bundle.

Toggling Autoplay on either Theme or Track will start music playback on game start.

Toggling rules:

- Track (The track and the theme starts playing syncronized)

- Theme (Will only play the theme and no music)


Inspector
Reactional Music
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Reactional Manager (Script)
Script
ReactionalManager
Default Section
Bundles
2
Main Out
None (Audio Mixer Group)
Basic Playback (Script)
Script
BasicPlayback
Autoplay Theme
Autoplay Track
Add Component

Best Practices

This document provides a few pointers on how to get the most out of the Reactional Plugin in Unity. It is not a complete guide but rather a collection of tips and tricks.

Summary

- Version Support
- Considerations
- Build Support

Version Support

- Unity 6
- Unity 2022 LTS
- Unity 2021 LTS
- Unity 2020 LTS

Considerations

When working in Unity, the Reactional API is made to be easy to use and prototype with. Depending on the size of your project and the complexity of the logic, consider this:

Simple Logic, A Player Prefab or just a Few Reactional dependent Objects in the Scene

When using Reactional API to react on gameplay, whether the thing reacting is a Player Prefab, or a Few Object Instances in the Scene, and / or if they are dependent on simple conditions, you can drive your logic locally in any Prefab without significant impact on performance.

Big projects, lots of Objects and complex logic

When using the Reactional API to drive complex behaviors of the gameplay, or using the same logic on multiple actors in a Scene, consider the consistency, scalability, and maintainability aspects. Centralizing Reactional logic into a manager could help to manage your performance costs.

Using a central ReactionalManager script on a single gameObject converted to a Prefab in the Hierarchy to handle logic calculations reduces computational overhead. For instance, rather than having each GameObject handle its own logic in complex scenes with many GameObjects or frequently duplicated objects, like pickups or other scene elements that are dependent on the Reactional API. Centralizing these calculations ensures they are computed only once, which optimizes performance.

Other Tips

- Heavy functions as LoadBundle should in most cases be called in Async mode.
- Use Object.DontDestroyOnLoad if you don't want to lose your Reactional Manager script values between scenes.

Build Support

- Win64
- macOS Intel

- macOS arm64
- Linux
- iOS
- Android

Check out the Build Settings for more information

Known Issues and Troubleshooting

Below are the known issues with the Reactional Plugin for Unity and their solutions. If you encounter an issue not listed here, please report it.

- Issue: Testbundles have an expiration time of 30 days and will not work in your project after that time.
    - Solution: Redownload your bundle via the **Platform**
- Issue: If you change the bundle, you have to click 'Reload Bundle'. Sometimes this will cause the game object to not fully update with the reloaded bundle after the change. This will make the game revert to the previous bundle.
    - Solution: You need to make any change on the Reactional game object, such as adding and removing something. Then save the game object to solve this issue.

Report Issue

If you find an issue, please let us know via the Support Section

Adding the Reactional SDK

Step 1. Download unitypackage

> **INFO**
>
> - Sign up at the platform and Download the latest version of the Unity Plugin **here**

Step 2. Installation

Open a new or existing Unity project and double click on the downloaded .unitypackage. A Unityprompt will ask you to confirm the files to be added to your project. Click to accept.

Step 3. You're set!

In your Assets folder you will find a folder called Reactional, containing plugin binaries as well as various scripts and demo content.

> **VERSION SUPPORT**
>
> - Unity 6
> - Unity 2022 LTS
> - Unity 2021 LTS
> - Unity 2020 LTS

> **PLATFORM SUPPORT**
>
> - Win64
> - macOS Intel
> - macOS arm64
> - Android
> - iOS

> 💡 Please reach out to us for console support.

Unity Changelog

Plugin Version 1.0rc11 - 24-07-01

- Fixed track fade. Load section changed to task. Load Playlist by section in case all called Default
- Added unity plugin version and fixed track repeat

Overview

The Reactional.Playback namespace deals with everything in regards to the system while it is active and playing.

It is divided into three parts:

- Theme
- Playlist
- MusicSystem

I'd advise to just quickly gloss over this page to get a basic sense of what is directly available. We'll explore these more in depth in the following chapters.

Setup

| Reactional.Setup.LoadBundle()
| Reactional.Setup.IsValid { get, set }
| Reactional.Setup.Samplerate { get, set }
| Reactional.Setup.Blocksize { get, set }
| Reactional.Setup.AllowPlay { get, set }
| Reactional.Setup.LoadBundles()
| Reactional.Setup.LoadBundle(string bundlename)
| Reactional.Setup.LoadSection(string section)
| Reactional.Setup.LoadPlaylist(string playlist)
| Reactional.Setup.LoadTrack(string track)
| Reactional.Setup.LoadTheme(string theme)

Playback.Theme

The theme, as you probably know by now, is the generative/interactive side of the Reactional Music system. Through the Playback.Theme static class you can easily access the functionalities of the loaded theme.

| Reactional.Playback.Theme.Play()
| Reactional.Playback.Theme.Stop()

| Reactional.Playback.Theme.GetThemeID()
| Reactional.Playback.Theme.GetTempo()
| Reactional.Playback.Theme.GetControls()
| Reactional.Playback.Theme.GetTriggers()
| Reactional.Playback.Theme.GetCurrentBeat()
| Reactional.Playback.Theme.GetNumParts()
| Reactional.Playback.Theme.GetCurrentPart()
| Reactional.Playback.Theme.GetPartName(int part)
| Reactional.Playback.Theme.GetPartOffset(int part)
| Reactional.Playback.Theme.GetPartDuration(int part)
| Reactional.Playback.Theme.GetNumBars()
| Reactional.Playback.Theme.GetCurrentBar()
| Reactional.Playback.Theme.GetBarOffset(int bar)
| Reactional.Playback.Theme.GetBarDuration(int bar)
| Reactional.Playback.Theme.Volume { get, set }
| Reactional.Playback.Theme.SetControl(string name, float value)
| Reactional.Playback.Theme.InstrumentOverride(int inst, float pulse, float pitch, float velocity, bool active, bool legato)

Playback.Playlist

The Playback.Playlist static class is focused on the audio track side of things, and is used to control playback of audio as well as playlist functionality. These functions are designed to work much like you would imagine a music player would.

| Reactional.Playback.Playlist.Play()
| Reactional.Playback.Playlist.PlayTrack(string, int, TrackInfo)
| Reactional.Playback.Playlist.PlayTrackByID(int trackID)
| Reactional.Playback.Playlist.Stop(float fadeoutTime)
| Reactional.Playback.Playlist.Next(bool autoplay = true)
| Reactional.Playback.Playlist.Prev(bool autoplay = true)
| Reactional.Playback.Playlist.Random(bool autoplay = true)
| Reactional.Playback.Playlist.IsLoaded()
| Reactional.Playback.Playlist.GetTrackID()
| Reactional.Playback.Playlist.GetCurrentTrackInfo()
| Reactional.Playback.Playlist.GetCurrentTrackMetadata()
| Reactional.Playback.Playlist.GetSelectedTrackIndex()

| Reactional.Playback.Playlist.Volume { get, set }
| Reactional.Playback.Playlist.IsPlaying { get, set }

Playback.MusicSystem

For nitty-gritty musical information and overarching controls, you can use the Playback.MusicSystem class.

| Reactional.Playback.MusicSystem.GetEngine()
| Reactional.Playback.MusicSystem.GetNextBeatAbsolute(float division)
| Reactional.Playback.MusicSystem.GetNextBeat(float division)
| Reactional.Playback.MusicSystem.GetCurrentBeat()
| Reactional.Playback.MusicSystem.GetCurrentMicroBeat()
| Reactional.Playback.MusicSystem.GetTempoBpm()
| Reactional.Playback.MusicSystem.BeatsToSeconds()
| Reactional.Playback.MusicSystem.WaitForNextBeat(float division)

Setup

Properties

Samplerate

- **Type**: double
- **Summary**: Get or Set the sample rate of the engine
- **Description**: Represents the sample rate at which the engine operates. Adjusting the sample rate may affect the quality and performance of audio processing.
- **Example**:

```
// Setting the Samplerate
Reactional.Setup.Samplerate = 44100;

// Getting the Samplerate
double rate = Reactional.Setup.Samplerate;
```

BlockSize

- **Type**: int
- **Summary**: Get or Set the block size of the engine
- **Description**: Represents the block size of the engine. The block size may affect the engine's processing capabilities, with larger blocks potentially offering better performance at the cost of increased latency.
- **Example**:

```

```

```
// Setting the BlockSize
Reactional.Setup.BlockSize = 512;

// Getting the BlockSize
int blockSize = Reactional.Setup.BlockSize;
```

Lookahead

- **Type**: int
- **Description**: Delays audio to be able to processing engine events ahead of time
- **Example**:

```
// Setting the Lookahead
Reactional.Setup.Lookahead = 1;

// Getting the Lookahead
int blockSize = Reactional.Setup.Lookahead;
```

IsValid

- **Type**: bool
- **Description**: Checks the validity of the `ReactionalManager` instance in the scene. Returns `true` if a valid instance exists; otherwise, it logs an error and returns `false`.
- **Example**:

```
bool valid = Reactional.Setup.IsValid;
```

AllowPlay

- **Type**: bool

- **Description**: Determines whether the `ReactionalEngine` instance is allowed to play. When true, the system is allowed to Process and Play audio. When false, the system is halted and no audio is played.
- **Example**:

```
// Enabling play
Reactional.Setup.AllowPlay = true;

// Checking if play is allowed
bool canPlay = Reactional.Setup.AllowPlay;
```

Methods

InitAudio

- **Return Type**: void
- **Parameters**: int bufferSize = -1, int sampleRate = -1
- **Description**: Initializes the audio settings for the engine. If no values are provided for `bufferSize` and `sampleRate`, it fetches the default values using `AudioSettings`.

UpdateBundles

- **Return Type**: void
- **Description**: Reads all bundles and populates the ReactionalManager in the inspector. Useful for Runtime reading of bundles added post release.

LoadBundles

- **Return Type**: async void
- **Description**: Loads all asset bundles found in the StreamingAssets folder.

LoadBundle

- **Return Type**: async void
- **Description**: Loads a specific asset bundle from the StreamingAssets folder. This is a convenience method; the first theme and playlist of the first bundle will be selected by default. Automatically loads all Sections in bundle.
- **Example**:

```
await Reactional.Setup.LoadBundle("MyMusicProject_12030");
```

LoadSection

- **Return Type**: async void
- **Description**: Loads a specific section from a specific asset bundle from the StreamingAssets folder. This is a convenience method; the first theme and playlist of said section will be selected by default. Automatically loads all Themes and Playlists in section.
- **Example**:

```
await Reactional.Setup.LoadSection("MyMusicProject_12030", "Level2");
```

LoadPlaylist

- **Return Type**: async Task
- **Description**: Loads a specific playlist from a specific asset bundle from the StreamingAssets folder. Automatically loads all Tracks.
- **Overloads**:

```
LoadPlaylist(string bundleName, string sectionName, string playlistName)
LoadPlaylist(string bundleName, string playlistName)
LoadPlaylist(int bundleIndex, string playlistName)
LoadPlaylist(string playlistName = "")
```

- **Example**:

```
await Reactional.Setup.LoadPlaylist("MyMusicProject_12030", "DefaultPlaylist");
```

LoadTrack

- **Return Type**: async void
- **Description**: Loads a specific track from a specific asset bundle from the StreamingAssets folder.
- **Example**:

```
await Reactional.Setup.LoadTrack("MyMusicProject_12030", "Summertime is sometimes
hot");
```

LoadTheme

- **Return Type**: async Task
- **Description**: Loads a specific track from a specific asset bundle from the StreamingAssets folder.
- **Overloads**:

```
LoadTheme(string bundleName, string sectionName, string themeName)
LoadTheme(string themeName = "")
```

- **Example**:

```
await Reactional.Setup.LoadTheme("Epic Encounters");
```

Enumerators

LoadType

- **Values**:

```
LoadInBackground  = 0
Synchronous       = 1
```

- **Description**: Selecting load type for certain methods.

UpdateMode

- **Values**:

```
Main        = 0
Threaded    = 1
AudioThread = 2
```

- **Description**: Selecting which thread the engine runs on.

AudioOutputMode

- **Values**:

```
Unity   = 0
Custom  = 1
```

- **Description**: Designates if Unity's default audio output setup will be in use.

Playback.Theme

Methods

GetThemeID

- **Return Type**: `int`
- **Description**: Retrieves the ID of the current theme.
- **Example**:

```
int themeID = Playback.Theme.GetThemeID();
```

IsLoaded

- **Return Type**: `bool`
- **Description**: Checks if a theme is currently loaded.
- **Example**:

```
bool loaded = Playback.Theme.IsLoaded();
```

Play

- **Return Type**: `void`
- **Description**: Begins the playback of the current theme. If a track is already loaded in the playlist, it copies the pre-roll settings to the theme before starting.
- **Example**:

```
Playback.Theme.Play();
```

Reset

- **Return Type**: `void`
- **Description**: Resets the current theme track.
- **Example**:

```
Playback.Theme.Reset();
```

Stop

- **Return Type**: `void`
- **Description**: Stops the playback of the current theme.
- **Example**:

```
Playback.Theme.Stop();
```

State

- **Return Type**: `int`
- **Description**: Retrieves the current state of the theme playback.
- **Example**:

```
int currentState = Playback.Theme.State();
```

GetControls

- **Return Type**: `Dictionary<string, (float, string)>`
- **Parameters**: bool istheme = true, Engine engine = null
- **Description**: Retrieves a list of control names associated with the theme or track, depending on the value of `istheme`.
- **Example**:

```
var (controlName, (controlValue, controlType)) = Playback.Theme.GetControls();
```

CurrentBeat

- **Return Type**: `float`
- **Description**: Gets the current beat value of the theme.
- **Example**:

```
float beat = Playback.Theme.CurrentBeat();
```

TempoBpm

- **Return Type**: `float`
- **Description**: Retrieves the tempo (in BPM) of the current theme.
- **Example**:

```
float bpm = Playback.Theme.TempoBpm();
```

SetControl

- **Return Type**: `void`
- **Parameters**: string controlName, float value = 1f, bool istheme = true
- **Description**: Sets the value of a specified control for the theme. This is the most important method as the control macros can be set up to fully control all of the music.
- **Example**:

```
Playback.Theme.SetControl("MyControl", 0.8f);
```

GetControl

- **Return Type**: `double`
- **Parameters**: string controlName, bool istheme = true
- **Description**: Retrieves the value of a specified control for the theme or track.
- **Example**:

```
double controlValue = Playback.Theme.GetControl("MyControl");
```

GetNumParts

- **Return Type**: `int`
- **Description**: Retrieves the number of parts in theme.
- **Example**:

```
int numOfParts = Playback.Theme.GetNumParts();
```

GetCurrentPart

- **Return Type**: `int`
- **Description**: Retrieves the current part in theme.
- **Example**:

```
int part = Playback.Theme.GetCurrentPart();
```

GetPartName

- **Return Type**: `string`
- **Parameters**: int part
- **Description**: Retrieves the name of the part with the correlated id.
- **Example**:

```
string partName = Playback.Theme.GetPartName(0);
```

GetPartOffset

- **Return Type**: `float`
- **Parameters**: `int` part
- **Description**: Retrieves the offset of the part with the correlated id.
- **Example**:

```
float offset = Playback.Theme.GetPartOffset(0);
```

GetPartDuration

- **Return Type**: `float`
- **Parameters**: `int` part
- **Description**: Retrieves the duration of the part with the correlated id.
- **Example**:

```
float duration = Playback.Theme.GetPartDuration();
```

GetNumBars

- **Return Type**: `int`
- **Description**: Retrieves the numbers of bars in the loaded theme.
- **Example**:

```
int numBars = Playback.Theme.GetNumBars();
```

GetCurrentBar

- **Return Type**: `int`
- **Description**: Retrieves the current bars in the loaded theme.
- **Example**:

```
int currentBar = Playback.Theme.GetCurrentBar();
```

GetBarOffset

- **Return Type**: `float`
- **Parameters**: `int` bar
- **Description**: Retrieves the offset of the bar with the id of variable 'bar' in the loaded theme.
- **Example**:

```
int currentBar = Playback.Theme.GetCurrentBar();
float offset = Playback.Theme.GetBarOffset(currentBar);
```

GetBarDuration

- **Return Type**: `float`
- **Parameters**: `int` bar
- **Description**: Retrieves the duration of the bar with the id of variable 'bar' in the loaded theme.
- **Example**:

```
int currentBar = Playback.Theme.GetCurrentBar();
float duration = Playback.Theme.GetBarDuration(currentBar);
```

TriggerStinger

- **Return Type**: `void`
- **Overloads**:

```
TriggerStinger(string stingerName, MusicSystem.Quant quant = MusicSystem.Quant.Half)
TriggerStinger(string stingerName, float value)
```

- **Description**:
- **Example**:

```
Playback.Theme.TriggerStinger("neutral, medium");
```

InstrumentOverride

- **Return Type**: `void`
- **Parameters**: int instrument, float pulseRate, float pitch, float velocity, bool active, int legato
- **Description**: Directly overrides and controls an instrument of the Interactive Theme. Can be used to give players direct influence over the soundtrack. PulseRate is how often a note trigger happens (1/1 at 0, 128th at 1). Pitch is a relative pitch position in the instruments defined range. Velocity is loudness/energy, Active sets the override to active or not. Legato defines if movement is needed to trigger new note.
- **Example**:

```
Playback.Theme.InstrumentOverride(0, 0.5f, 0.6f, 1f, true, 1);
```

GetOverridableInstruments

- **Return Type**: `List<string>`
- **Description**: Returns all overridable instrument from the loaded theme.
- **Example**:

```
var instruments = Playback.Theme.GetOverridableInstruments();
```

Properties

Volume

- **Type**: `float`
- **Description**: Get or set the volume (gain) for the theme.
- **Example**:

```
// Setting the volume
Playback.Theme.Volume = 0.5f;

// Getting the volume
float volume = Playback.Theme.Volume;
```

Playback.Playlist

Methods

CurrentBeat

- **Return Type**: float
- **Description**: Retrieves the current beat value of the playlist's track.
- **Example**:

```
float beat = Playback.Playlist.CurrentBeat();
```

TempoBpm

- **Return Type**: float
- **Description**: Retrieves the tempo (in BPM) of the track in the playlist.
- **Example**:

```
float bpm = Playback.Playlist.TempoBpm();
```

GetTrackID

- **Return Type**: int
- **Description**: Retrieves the ID of the current track in the playlist.
- **Example**:

```
int trackID = Playback.Playlist.GetTrackID();
```

GetCurrentTrackMetadata

- **Return Type**: json string
- **Description**: Dumps some data on the current track, such as artist, title bpm, duration etc.
- **Example**:

```
Debug.Log(Reactional.Playback.Playlist.GetCurrentTrackMetadata())
```

GetCurrentTrackInfo

- **Return Type**: TrackInfo
- **Description**: Return a TrackInfo object on the current selected track.
- **Example**:

```
TrackInfo selectedTrack = Reactional.Playback.Playlist.GetCurrentTrackInfo()
```

GetSelectedTrackIndex

- **Return Type**: int
- **Description**: Retrieves the index of the track currently selected/playing, from the Loaded Tracks list.
- **Example**:

```
int selectedTrack = Playback.Playlist.GetSelectedTrackIndex();
```

IsLoaded

- **Return Type**: bool
- **Description**: Checks if a track is currently loaded in the playlist.
- **Example**:

```
bool isLoaded = Playback.Playlist.IsLoaded();
```

PlayTrack

- **Return Type**: void
- **Overloads**:

```
PlayTrack(TrackInfo ti, float fadeintime = 0, float fadeouttime = 0.1f)
PlayTrack(int loadedTrackIndex, float fadeintime = 0, float fadeouttime = 0.1f)
PlayTrack(string trackName, float fadeintime = 0, float fadeouttime = 0.1f)
```

- **Description**: Retrieves the ID of the current track in the playlist.
- **Example**:

```
Playback.Playlist.PlayTrack();
```

PlayTrackByID

- **Return Type**: async void
- **Description**: Play a track based on the music system assigned ID.
- **Example**:

```

```

```
Playback.Playlist.PlayTrackByID(int ID);
```

Next

- **Return Type**: void
- **Parameters**: float fadeouttime = 0.1f, float fadeintime = 0f, bool autoplay = true
- **Description**: Moves to the next track in the playlist.
- **Example**:

```
Playback.Playlist.Next();
```

Prev

- **Return Type**: void
- **Parameters**: float fadeouttime = 0.1f, float fadeintime = 0f, bool autoplay = true
- **Description**: Moves to the previous track in the playlist.
- **Example**:

```
Playback.Playlist.Prev();
```

Random

- **Return Type**: void
- **Parameters**: float fadeouttime = 0.1f, float fadeintime = 0f, bool autoplay = true
- **Description**: Loads a random track from the playlist.

- **Example**:

```
Playback.Playlist.Random();
```

GetState

- **Return Type**: void
- **Description**: Retrieves the current state of the playlist's playback as a MusicSystem.PlaybackState
- **Example**:

```
MusicSystem.PlaybackState state = Playback.Playlist.GetState();
if (state == Playing) then ...
```

Stop

- **Return Type**: void
- **Parameters**: float fadeout = 0
- **Description**: Stops the playback of the track in the playlist.
- **Example**:

```
Playback.Playlist.Stop(0.5f);
```

Play

- **Return Type**: void
- **Description**: Starts playback of the current track in the playlist. If no track is specified, it attempts to load a default track or the first available track.

- **Example**:

```
Playback.Playlist.Play();
```

FadeOut

- **Return Type**: void
- **Parameters**: float fadeout
- **Description**: Fades out the current playing track.
- **Example**:

```
Playback.Playlist.FadeOut(1f);
```

FadeIn

- **Return Type**: void
- **Parameters**: float fadein
- **Description**: Fades in the current playing track.
- **Example**:

```
Playback.Playlist.FadeIn(1f);
```

GetNumParts

- **Return Type**: `int`
- **Description**: Retrieves the number of parts in theme.

- **Example**:

```
int numOfParts = Playback.Playlist.GetNumParts();
```

GetCurrentPart

- **Return Type**: `int`
- **Description**: Retrieves the current part in track.
- **Example**:

```
int part = Playback.Playlist.GetCurrentPart();
```

GetPartName

- **Return Type**: `string`
- **Parameters**: int part
- **Description**: Retrieves the name of the part with the correlated id.
- **Example**:

```
string partName = Playback.Playlist.GetPartName(0);
```

GetPartOffset

- **Return Type**: `float`
- **Parameters**: `int` part
- **Description**: Retrieves the offset of the part with the correlated id.

- **Example**:

```
float offset = Playback.Playlist.GetPartOffset(0);
```

GetPartDuration

- **Return Type**: `float`
- **Parameters**: `int` part
- **Description**: Retrieves the duration of the part with the correlated id.
- **Example**:

```
float duration = Playback.Playlist.GetPartDuration();
```

GetNumBars

- **Return Type**: `int`
- **Description**: Retrieves the numbers of bars in the loaded track.
- **Example**:

```
int numBars = Playback.Playlist.GetNumBars();
```

GetCurrentBar

- **Return Type**: `int`
- **Description**: Retrieves the current bars in the loaded track.
- **Example**:

```
int currentBar = Playback.Playlist.GetCurrentBar();
```

GetBarOffset

- **Return Type**: `float`
- **Parameters**: `int` bar
- **Description**: Retrieves the offset of the bar with the id of variable 'bar' in the loaded track.
- **Example**:

```
int currentBar = Playback.Playlist.GetCurrentBar();
float offset = Playback.Playlist.GetBarOffset(currentBar);
```

GetBarDuration

- **Return Type**: `float`
- **Parameters**: `int` bar
- **Description**: Retrieves the duration of the bar with the id of variable 'bar' in the loaded track.
- **Example**:

```
int currentBar = Playback.Playlist.GetCurrentBar();
float duration = Playback.Playlist.GetBarDuration(currentBar);
```

Properties

Volume

- **Type**: float
- **Description**: Get or set the volume (gain) for the track in the playlist.
- **Example**:

```
// Setting the volume
Playback.Playlist.Volume = 0.5f;

// Getting the volume
float volume = Playback.Playlist.Volume;
```

IsPlaying

- **Type**: boolean
- **Description**: Returns true if a track is playing.
- **Example**:

```
if (Reactional.Playback.Playlist.IsPlaying) then ...
```

State

- **Return Type**: int
- **Description**: Retrieves the current state of the playlist's playback.
- **Example**:

```
int state = Playback.Playlist.State;
```

MusicSystem

Methods

ScheduleAudio

- **Return Type**: void
- **Parameters**: AudioSource audioSource, float quant = 1, float timeOffset = 0
- **Description**: Schedules an audio source to play at a specified quantization time offset.
- **Example**:

```
MusicSystem.ScheduleAudio(myAudioSource, 2, 0.5f);
```

GetEngine

- **Return Type**: Engine
- **Description**: Retrieves the current musical engine instance.
- **Example**:

```
Engine engine = MusicSystem.GetEngine();
```

GetTimeToBeat

- **Return Type**: float
- **Parameters**: float quant, float offset = 0, bool theme = true

- **Description**: Gets the number of beats left until the next quantized beat.
- **Example**:

```
float timeToBeat = MusicSystem.GetTimeToBeat(4);
```

GetRealTimeToBeat

- **Return Type**: float
- **Parameters**: float quant
- **Description**: Converts the time to beat into real time duration.
- **Example**:

```
float realTime = MusicSystem.GetRealTimeToBeat(4);
```

GetNextBeat

- **Return Type**: float
- **Parameters**: float quant, float offset = 0, bool theme = true
- **Description**: Retrieves the beat position of the next quantized beat. A quantized beat means the next occurence of said multiple. A value of "4" will return the next beat multiple of 4; 0, 4, 8, 12 etc. A value of 1 will return every rounded beat.
- **Example**:

```
float nextBeat = MusicSystem.GetNextBeat(4);
```

GetNextBeatAbsolute

- **Return Type**: long
- **Parameters**: float quant, float offset = 0, bool theme = true
- **Description**: Returns the absolute value of the next quantized beat.
- **Example**:

```
long absoluteBeat = MusicSystem.GetNextBeatAbsolute(4);
```

GetCurrentBeat

- **Return Type**: float
- **Description**: Retrieves the current beat position.
- **Example**:

```
float currentBeat = MusicSystem.GetCurrentBeat();
```

GetCurrentMicroBeat

- **Return Type**: double
- **Description**: Retrieves the current beat position with micro precision.
- **Example**:

```
double microBeat = MusicSystem.GetCurrentMicroBeat();
```

GetTempoBpm

- **Return Type**: float

- **Description**: Retrieves the tempo in beats per minute.
- **Example**:

```
float bpm = MusicSystem.GetTempoBpm();
```

BeatsToSeconds

- **Return Type**: float
- **Parameters**: float beats
- **Description**: Converts beat value into seconds based on the current BPM.
- **Example**:

```
float seconds = MusicSystem.BeatsToSeconds(4);
```

DuckMusic

- **Return Type**: IEnumerator
- **Parameters**: float amp
- **Description**: Gradually reduces the music volume by a specified amplitude.
- **Example**:

```
StartCoroutine(MusicSystem.DuckMusic(0.5f));
```

Properties

PlaybackAllowed

- **Type**: bool
- **Description**: Get or set whether playback is allowed.
- **Example**:

```
// Setting PlaybackAllowed
MusicSystem.PlaybackAllowed = true;

// Getting PlaybackAllowed
bool isAllowed = MusicSystem.PlaybackAllowed;
```

Enumerators

Quant

- **Values**:

```
Whole     = 1
Half      = 2
Quarter   = 4
Eighth    = 8
Sixteenth = 16
```

- **Description**: Represents musical quantization values.

PlaybackState

- **Values**:

```
Stopped = 0
Playing = 1
Paused  = 2
```

- **Description**: States of music playback.

Classes

WaitForNextBeat

- **Description**: Custom yield instruction that waits for the next quantized beat.
- **Example**:

```
yield return new MusicSystem.WaitForNextBeat(4);
```

Engine

ReactionalEngine

Via the Engine you have the opportunity to subscribe to delegates functions, that will trigger on events. This has an added benefit in that we can listen in on said communication, and also communicate directly with the various features, as a sort of low level API. The most common use case would be to listen for "noteon", and "noteoff" messages generated by the engine.

RouteNoteOn

- **Return Type**: void
- **Summary**: Subscribes to the callback for Note On events.
- **Description**: Invoked when a "noteon" message is received from the Reactional Engine. This callback processes note-on events, providing details such as the event offset, sink index, lane index, pitch, and velocity.
- **Parameters**:
- **offset (double)**: The time offset in microbeats when the event occurs.
- **sink (int)**: The sink index where the event originated.
- **lane (int)**: The lane or output group index.
- **pitch (float)**: The pitch of the note.
- **velocity (float)**: The velocity of the note.
- **Example**:

```
// Subscribe to the Note On event
  onNoteOn += RouteNoteOn;

  // Example callback implementation
```

```
  void RouteNoteOn(double offset, int sink, int lane, float pitch, float velocity)
  {
  // Do Stuff
  }
```

RouteNoteOff

- **Return Type**: void
- **Summary**: Subscribes to the callback for Note Off events.
- **Description**: Invoked when a "noteoff" message is received from the Reactional Engine. This callback processes note-off events, providing details such as the event offset, sink index, lane index, pitch, and velocity.
- **Parameters**:
- **offset (double)**: The time offset in microbeats when the event occurs.
- **sink (int)**: The sink index where the event originated.
- **lane (int)**: The lane or output group index.
- **pitch (float)**: The pitch of the note.
- **velocity (float)**: The velocity of the note.
- **Example**:

```
// Subscribe to the Note Off event
  onNoteOff += RouteNoteOff;

  // Example callback implementation
  void RouteNoteOff(double offset, int sink, int lane, float pitch, float velocity)
  {
  // Do Stuff
  }
```

RouteStingerEvent

- **Return Type**: void
- **Summary**: Subscribes to the callback for Stinger events.
- **Description**: Invoked when a "stinger/start" or "stinger/stop" message is received from the Reactional Engine. This callback handles stinger events, providing the time offset, whether it's a start or stop event, and the stinger's origin.
- **Parameters**:
- **offset (double)**: The time offset in microbeats when the event occurs.
- **startevent (bool)**: Indicates whether the event is a start or stop of the stinger.
- **stingerOrigin (int)**: The origin of the stinger event.
- **Example**:

```
// Subscribe to the Stinger event
  stingerEvent += RouteStingerEvent;

  // Example callback implementation
  void RouteStingerEvent(double offset, bool startevent, int stingerOrigin)
  {
  // Do Stuff
  }
```

RouteBarBeat

- **Return Type**: void
- **Summary**: Subscribes to the callback for Bar and Beat events.
- **Description**: Invoked when a "bar" message is received from the Reactional Engine. This callback processes bar and beat events, providing the time offset, bar number, and beat number.
- **Parameters**:

- **offset (double)**: The time offset in microbeats when the event occurs.
- **bar (int)**: The bar number in the current sequence.
- **beat (int)**: The beat number within the bar.
- **Example**:

```
// Subscribe to the Bar and Beat event
  onBarBeat += RouteBarBeat;

  // Example callback implementation
  void RouteBarBeat(double offset, int bar, int beat)
  {
  // Do Stuff
  }
```

RouteAudioEnd

- **Return Type**: void
- **Summary**: Subscribes to the callback for Audio End events.
- **Description**: Invoked when an "audio/end" message is received from the Reactional Engine. This callback handles events that indicate the end of an audio sequence.
- **Parameters**: None.
- **Example**:

```
// Subscribe to the Audio End event
  onAudioEnd += RouteAudioEnd;

  // Example callback implementation
  void RouteAudioEnd()
  {
  // Do Stuff
  }
```
