RhythmTool 3.3 Documentation

Overview

RhythmTool is a straightforward scripting package for Unity, with all the basic functionality for creating games that react to music.

RhythmTool analyzes a song without the need of playing the song at the same time. It can analyze an entire song before playing it, or while it's being played. There are a number of types of data it provides:

- Beats
- Pitch
- Onsets
- Changes in overall intensity
- Volume

This data can be used in various ways and is provided through an easy to use asset and event system.

RhythmTool is designed to analyze and sync songs with a known length. Unfortunately it is not possible to analyze a continuous stream of data, like a web stream or mic input.

Contact

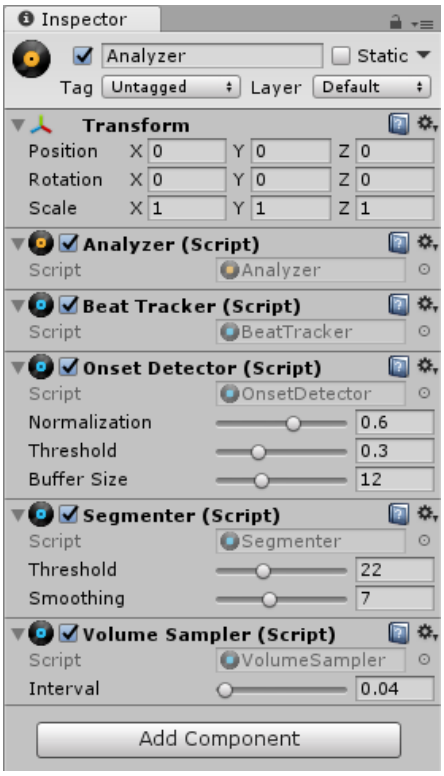
If you have any questions, suggestions, feedback or comments, please do one of the following:

- Send me an email: tim@hellomeow.net
- Make a post in the forum thread on the Unity forums

Analyzing a song

To start analyzing a song, first we need to create a RhythmAnalyzer. The RhythmAnalyzer is a component that takes an AudioClip, chops it up into small chunks and passes it to different analyses.

Now we can add different Analyses. An Analysis is a component which provides a specific set of information of the song. For example, the Beat Tracker will try to find the beat of the song and provides information such as beat locations and BPM. Here we can also configure the different Analysis settings. The default settings are tuned to give the best results for as many songs as possible.



After we have created a RhythmAnalyzer with a number of Analyses, we can right click an AudioClip and click **RhythmTool > Analyze**. This will start analyzing the song and create a RhythmData asset. This menu option will use the first available analyzer in the scene.

Alternatively, we can call RhythmAnalyzer.Analyze() to start analyzing an AudioClip from a script.

Analyzing a song takes several seconds. We can track the progress with RhythmAnalyzer.progress and isDone.

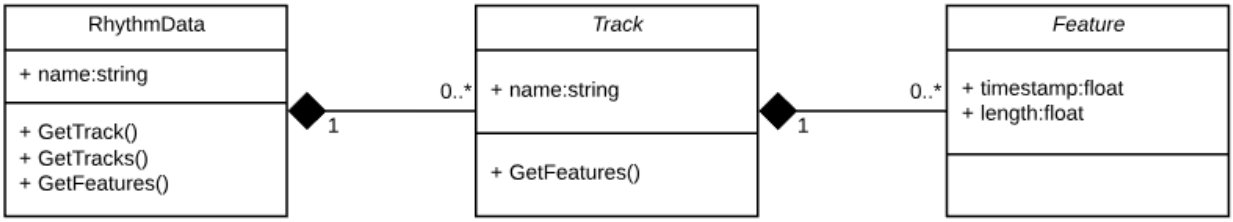
In most cases there is nothing wrong with using the RhythmData while the song is still being analyzed. However, it can take a few frames before an initial length of the song is analyzed. If you try to use data near the start of the song right away, it might not be available yet.

```
using UnityEngine;
using RhythmTool;

public class AnalyzeExample : MonoBehaviour
{
    public RhythmAnalyzer analyzer;

    public AudioClip audioClip;

    public RhythmData rhythmData;

    void Start ()
    {
        //Start analyzing a song.
        rhythmData = analyzer.Analyze(audioClip);
    }
}
```

RhythmData

RhythmData has the following structure. A RhythmData object can have several Tracks, each of which contains a collection of a specific type of Feature. Feature is the base type of information in a song. Each Feature has a timestamp and a length.

Like mentioned before, each Analysis will provide a Track. For example, the Beat Tracker will provide a Track named Beats, which includes Beats. Beat derives from Feature and adds BPM information.



To find a specific Track, we can use RhythmData.GetTrack() or RhythmData.GetTracks() . We can also use RhythmData.GetFeatures() to directly find Features of a specific type or with a specific Track name.

```
using UnityEngine;
using RhythmTool;

public class AnalyzeExample : MonoBehaviour
{
    public Analyzer analyzer;

    public AudioClip audioClip;

    public RhythmData rhythmData;

    void Start ()
    {
        //Start analyzing a song.
        rhythmData = analyzer.Analyze(audioClip);

        //Find a track with Beats.
        Track<Beat> track = rhythmData.GetTrack<Beat>();
    }
}
```

Using analysis results

As mentioned before, the RhythmAnalyzer provides a RhythmData object with several tracks, each of which contains a collection of Features. Possible tracks are the following:

- Beats
  These represent the rhythm of the song.
- Onsets
  These represent the start of a note or sound.
- Chroma
  Chroma features are closely related to pitch and represent the most prominent notes at a certain time in the song.
- Segments
  These represent sections of the song at which large changes in average volume occur. These changes often indicate different segments of a song.
- Volume
  These represent the volume of the song at different points in time.

Synchronizing results

Features in a track each have a timestamp. Features can be found by looking within a certain time frame with Track.GetFeatures() or RhythmData.GetFeatures(). This fills a list with every feature that occurs within a certain time frame.

To find features for the part of the song that is currently being played, the best way is to keep track of an AudioSource's time. For example, finding out if a beat has been detected at the current time in the song can be done like this:

```
public RhythmData rhythmData;
public AudioSource audioSource;

private float prevTime;
private List<Beat> beats;

void Awake()
{
    beats = new List<Beat>();
}

...
```

```
void Update()
{
    //Get the current playback time of the AudioSource.
    float time = audioSource.timeSeconds;

    //Clear the list.
    beats.Clear();

    //Find all beats for the part of the song that is currently playing.
    rhythmData.GetFeatures<Beat>(beats, prevTime, time);

    //Do something with the Beats here.
    foreach(Beat beat in beats)
    {
        Debug.Log("A beat occurred at " + beat.timestamp);
    }

    //Keep track of the previous playback time of the AudioSource.
    prevTime = time;

}
```

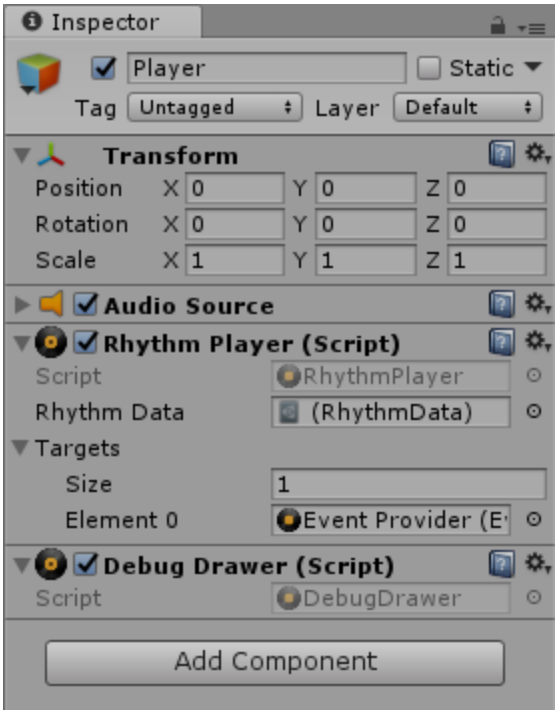
Using events

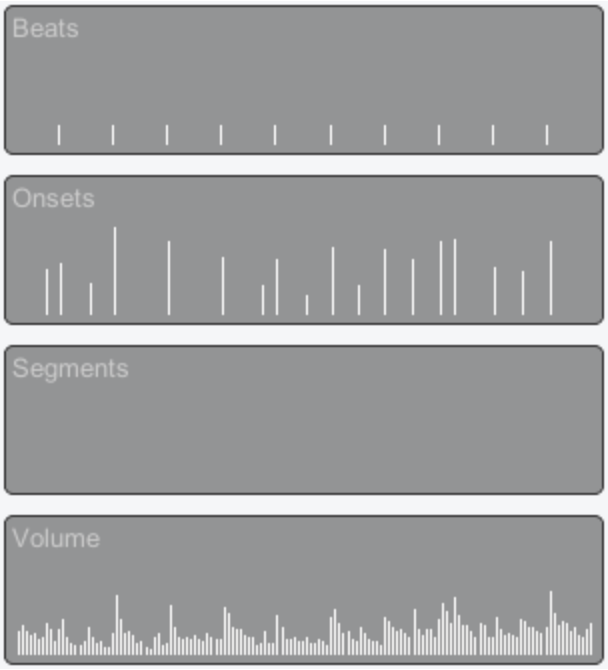
RhythmTool's event system consists of two parts. The RhythmPlayer and the RhythmEventProvider.

RhythmPlayer

The RhythmPlayer is a component that can be added next to an AudioSource.

When it is added next to an AudioSource, the RhythmPlayer will keep track of the time and send the time data and RhythmData to its targets. Here we can also add a component called the DebugDrawer. This component will draw a RhythmPlayer's RhythmData as the song is playing.

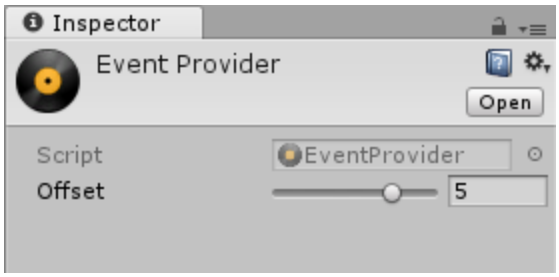




RhythmEventProvider

The RhythmEventProvider is an asset that can be created by clicking the right mouse button and clicking **Create > RhythmTool > EventProvider**. Once added to a RhythmPlayer, it will process the time data and RhythmData of the RhythmPlayer and trigger events.

It can also be given an offset in seconds, so events can be triggered ahead of time. For example, an offset of 5 will trigger events 5 seconds before they would occur.



Use RhythmEventProvider.Register() to subscribe methods to an event for a specific track or feature type. For example, to register a method to an event for the Beats Track:

```
public RhythmEventProvider eventProvider;

void Start()
{
    //Register the OnBeat method to the event provider.
    eventProvider.Register<Beat>(OnBeat);
}

private void OnBeat(Beat beat)
{
    //Do something when a Beat occurs.
    Debug.Log("A beat occurred at " + beat.timestamp);
}
```

When using events in C# with Unity, make sure to unsubscribe from events for Components that are destroyed. Otherwise the subscribed method will still be called even if the Component is destroyed.

```
void OnDestroy()
{
    eventProvider.Unregister<Beat>(OnBeat);
}
```

Importing songs at runtime

Songs can be imported at runtime with a separate, free package named AudioImporter. AudioImporter loads an audio file into an AudioClip. The package includes a simple file browser that can be used to select a file.

The package can be found on the Asset Store and on Github.

To import a song, we need an AudioImporter component. We can add NAudioImporter (or MobileImporter for Android or iOS) to a GameObject and start using it.

To start importing a file, use AudioImporter.Import(). This will start loading the file into an AudioClip. This happens in the background as much as possible. Use the AudioImporter.Loaded event to obtain the audioclip. Alternatively you can use AudioImporter.isDone or AudioImporter.progress to find out when the AudioClip is available.

The AudioClip can then be passed on to a RhythmAnalyzer. See the documentation for AudioImporter for more information.

```
using UnityEngine;
using RhythmTool;

public class ImporterExample : MonoBehaviour
{
    public string path;
    public AudioImporter importer;
    public RhythmAnalyzer analyzer;

    void Awake()
    {
        importer.Loaded += OnLoaded;
        importer.Import(path);
    }

    private void OnLoaded(AudioClip clip)
    {
        analyzer.Analyze(clip);
    }
}
```

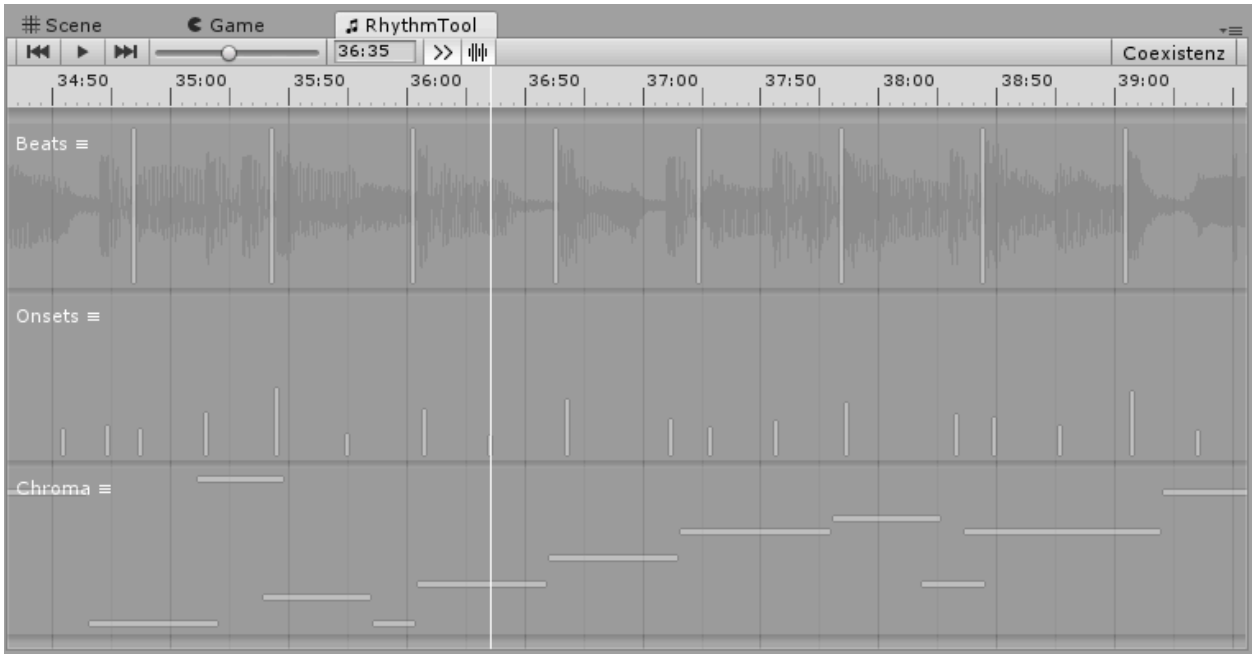
RhythmTool Editor

The RhythmTool editor can be used to inspect and edit analyzed songs as well as create RhythmData assets with custom tracks.

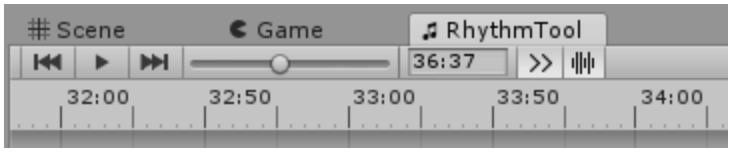
Using the Editor

To begin using the editor, simply double-click a RhythmData asset. The editor can also be opened by going to **Window > RhythmTool**.

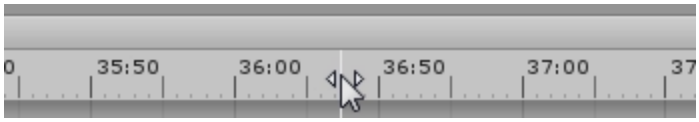
This will open a new window with the editor. The editor window shows the different Tracks and Features contained in the RhythmData object.



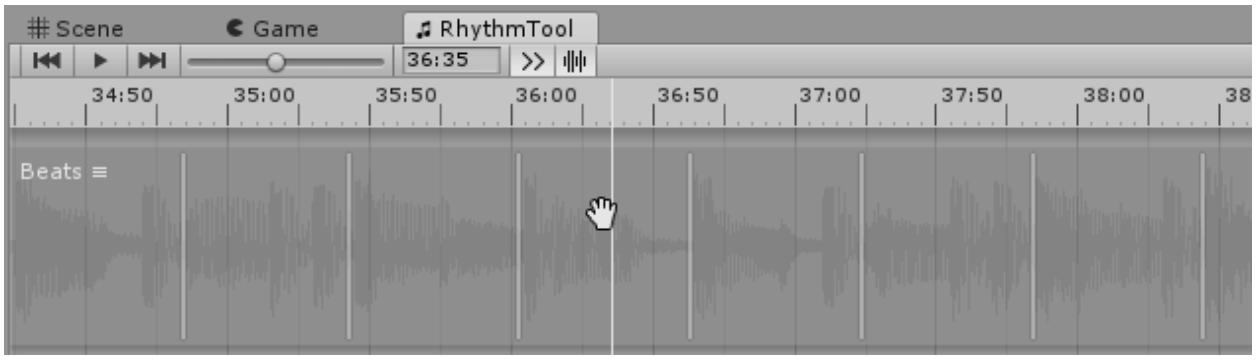
The toolbar at the top of the editor has basic playback controls like play, pause and volume. Here you can also find options to enable auto-scrolling and to enable or disable the waveform preview.



To set the current playback time, click anywhere in the timeline at the top of the editor. The timeline also lets you scrub around in the song by clicking and dragging.

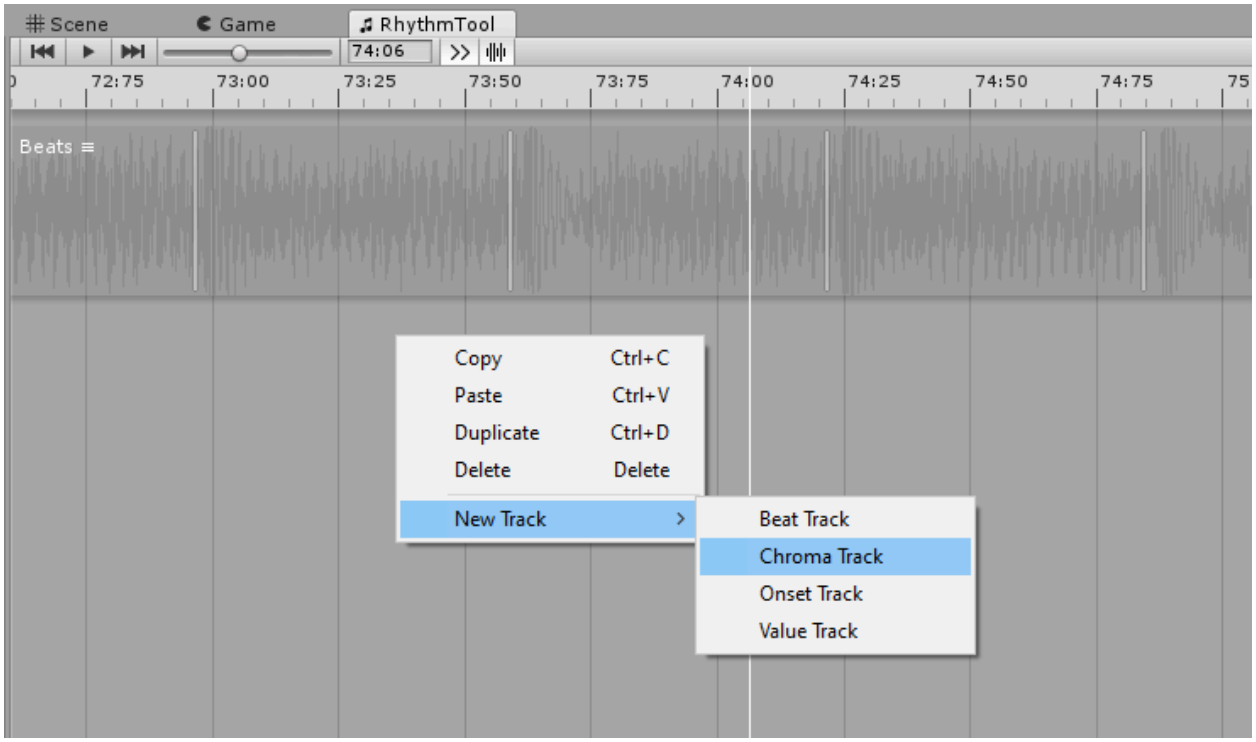


Use the middle mouse button to pan around and use the scroll wheel to zoom.



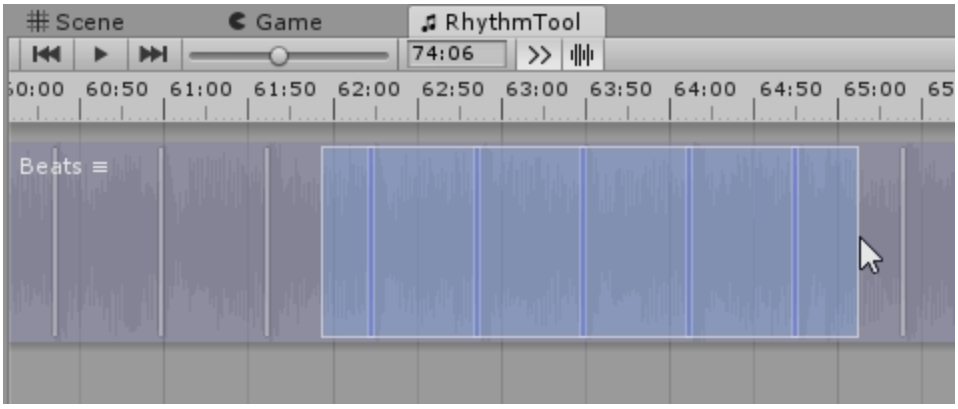
Editing Tracks

Right click anywhere in the editor window to open the context menu. The context menu lets you add, remove, copy and paste Tracks.



New features can be created by double clicking anywhere on a Track, or by using the context menu on a Track. Select Features by clicking on them, or by using a selection box on a Track.

Selected Features can be moved around by dragging them around in the editor, or by editing their timestamp directly in the inspector.



The inspector shows selected Features and their properties. The inspector has three different modes for editing the selected features.

Delta and **Absolute** allow for multi-editing Features, while the **List** editing mode will show all Features individually. Delta will edit the properties of all selected features based on the difference or delta of the modified property. Absolute will set the properties of all selected Features to the same value.