Unity Time Rewinder documentation (ver 1.0)

Unity Time Rewinder lets you rewind defined object states and move freely on time axis. It supports custom trackers with custom variable tracking that is easy to setup. System uses highly efficient circular buffer implementation to store and retrieve the values.

Downloading Time Rewinder and importing to custom Unity Project

You can either download the whole Unity example project from Github or download the prepared Unity package on Github page in releases. To use the Time Rewinder, only TimeRewinderImplemention folder is required, but i highly suggest checking the demoscene with prepared examples.

Start using Time Rewinder in your own project

To start using Time Rewinder, each scene must contain **RewindManager.cs** script, that is essential. In TimeRewinderImplementation/Scripts folder, there are prepared two scripts. **RewindBySlider.cs** and **RewindByKeyPress.cs.** As names suggest, both scripts let you rewind time differently. You can choose between these two prepared solutions that best suit your needs, or you can also write your own input rewind logic (more on that in the last chapter of this documentation).

Showcases of rewinding with these two solutions are prepared in demoscenes TimeRewindExample 1 and TimeRewindExample 2 that are located in ExampleScene folder.

RewindManager.cs contains definition of how many seconds should be tracked, you can change this value, although provided Slider is setup with animations to 12 seconds by default, so you would have to also update the slider settings. In case you want to use **RewindByKeyPress.cs**, no additional setting similar to Slider is required and you can simply change the value in code to your liking.

```
/// <summary>
/// Variable defining how much into the past should be tracked, after set limit is hit, old values will be overwritten in circular buffer
/// </summary>
public readonly float howManySecondsToTrack = 12;
```

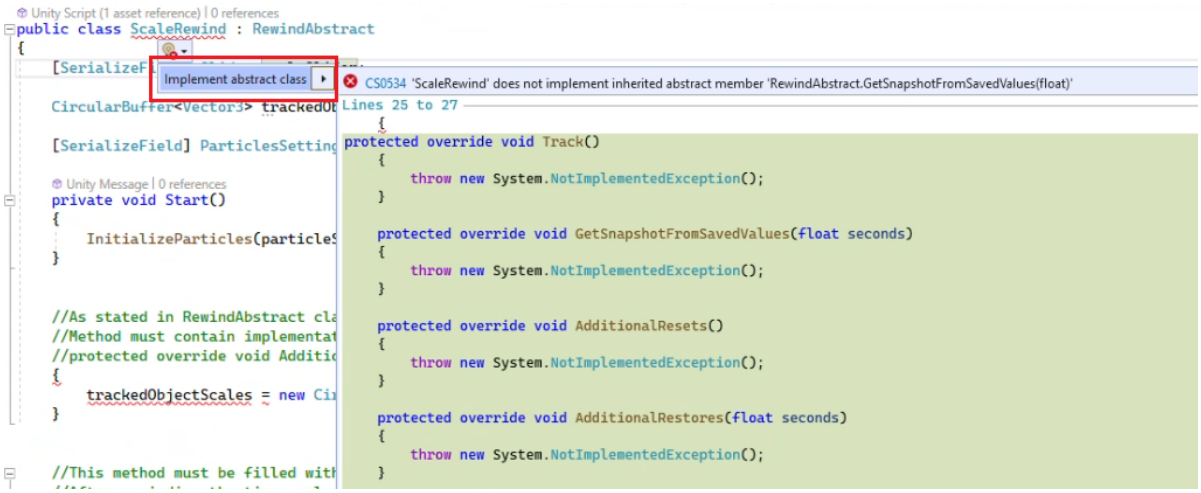
For straight from the box use, import one of the prefabs that you choose from TimeRewinderImplementation/Prefabs into the scene (these prefabs already contain **RewindManager.cs**). On desired object that you want to rewind you can use prepared script **GenericRewinds.cs**, that you attach to the object and then check what properties you want to track.

Currently straight from the box you can track object position, rotations, velocity, animator states, audio states and particle effects. Adding custom variable is easy, and i will show you how to do it in next section. To start rewinding, play the scene. If you choosen **RewindByKeyPress.cs**, you simply hold SpaceBar to rewind back. If you choosen **RewindBySlider.cs**, grab the slider handle and you can see, that you can move on time axis freely and see preview of historic snapshots. When you release the slider, active object values will be overwritten with preview values.

Adding custom tracker with custom variables tracking

In TimeRewinderImplementation/Scripts/ImplementedObjects, you can find **ScaleRewinds.cs** and **TimerRewinds.cs** which are two examples of implemented custom trackers. The first thing when doing custom tracker is implementing **RewindAbstract.cs** abstract class, which contains all needed

things for custom tracker. Simply inherit this class and let the editor implement all its methods.



To add custom variable tracking, add CircularBuffer (CircularBuffer is implemented in **CircularBuffer.cs**) with data structure you want to track. In our case, in Scale rewind, we want to track Vector3 values

```
//This script is showing setup of simple one custom variable tracking
Unity Script (1 asset reference) | 0 references
public class ScaleRewind : RewindAbstract
{
    [SerializeField] Slider scaleSlider;

    CircularBuffer<Vector3> trackedObjectScales;        //For storing data, use this CircularBuffer class
```

Now you can implement what values will be tracked and restored. I advise you to add 2 own separate methods for clarity (Tracking method and Rewind method). In these methods you simply tell what values will be tracked and then how it will be restored. Use **CircularBuffer.WriteLastValue()** and **CircularBuffer.ReadFromBuffer()** to write and restore from buffer positions

```
// This is an example of custom variable tracking
1 reference
public void TrackObjectScale()
{
    trackedObjectScales.WriteLastValue(transform.localScale);
}


// This is an example of custom variable restoring
1 reference
public void RestoreObjectScale(float position)
{
    transform.localScale = trackedObjectScales.ReadFromBuffer(position);

    //While we are at it, we can also additionaly restore slider value to match the object scale
    scaleSlider.value = transform.localScale.x;
}
```

Now define what you really want to track in **Track()** method override. You can tell it to track your own variables also with combination of already implemented tracking solutions (eg. Tracking position,rotation,velocity,animator…). This is the place where you should add your implemented tracking method.

```
//In this method define what will be tracked. In our case we want to track already implemented audio tracking,particle tracking + new custom added variable scale tracking
2 references
protected override void Track()
{
    TrackParticles();
    TrackAudio();
    TrackObjectScale();
}
```

Similarly to **Track()** method, you must also fill **GetSnapshotFromSavedValues()** method override, where is defined what will be restored on rewind. This is the place where you should add you own implemented rewind method.

```
//In this method define, what will be restored on time rewinding. In our case we want to restore previous audio state, particles + object scale state
3 references
protected override void GetSnapshotFromSavedValues(float seconds)
{
    float position = seconds * howManyRecordsPerSecond;        //For time rewinding purposes, use this calculation to restore correct position value (seconds*howManyRecordsPerSecond)

    RestoreParticles(position);
    RestoreAudio(position);
    RestoreObjectScale(position);
}
```

The last thing you have to do for your own custom variable rewinding is to fill **AditionalResets()** and **AdditionalRestores()** overrides method.

Additional reset method must contain implementation of reseting the custom buffers. Method is being triggered on actions like scene reload or **RewindManager.cs** reenabling, where in certain phase of the game you might want to turn off tracking and enable it later

Aditional restores must contain buffers restoration after rewind, to correctly set next buffer write position.

Implement these two methods similarly as shown in example here

```
//As stated in RewindAbstract class, this method must be filled with circular buffer that is used for custom variable tracking
//Method must contain implementation of reset of the circullar buffer to work correctly
2 references
protected override void AdditionalResets()
{
    trackedObjectScales = new CircularBuffer<Vector3>(howManyItemsFit);     //For time rewinding purposes, give the CircularBuffer constructor this variable (howManyItemsFit)
}


//After rewinding the time, values that were previously stored in buffer are obsolete.
//This method must contain implementation of moving bufferPosition for next write to correct position in regards to time rewind
2 references
protected override void AdditionalRestores(float seconds)
{
    trackedObjectScales.MoveLastBufferPosition(seconds * howManyRecordsPerSecond);  //For time rewinding purposes, use these attributes (seconds*howManyRecordsPerSecond)
}
```

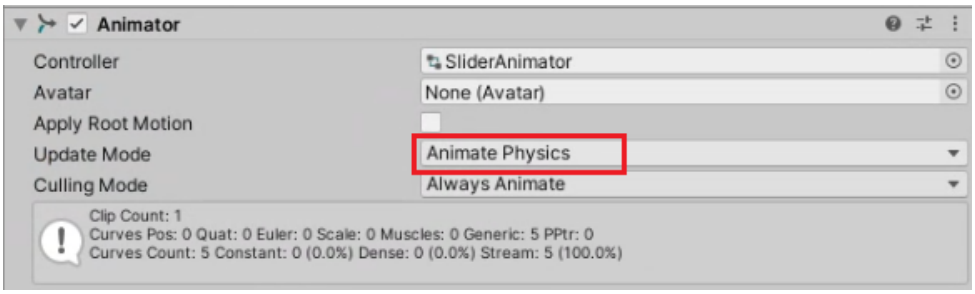
General mindset

Time rewind system is implemented with **RewindManager.cs,** that acts as main controller of all rewinds and scripts that implement **RewindAbstract.cs** abstract class that defaultly subscribes to **RewindManager.cs** events. Each tracked object that you want to track must contain script component, that inherits from **RewindAbstract.cs** class.

Custom tracker implementation is described in chapter above. Being component of specified object also means, that tracking cannot be done, when object that you want to track is disabled. So while rewinding time, do not disable objects you want to track. Use other techniques as for example moving the object from visible area or disable its mesh renderer. This rewind mindset applies to all kinds of tracking, except for Particles tracking, where you **can** disable specified particle systems. Particle system tracking is implemented slightly differently from other types of tracking, due to internal Unity reasons. Particle system caviats will be described in next chapter.

Tracking and rewinding is implemented with help of FixedUpdate(). I was experimenting with tracking intensity that you could set thru variable, but at the end i decided to stick with certainty that FixedUpdate() provides. Maybe in the future if requested i might look into it again, but for now all implementation is hardcoded with FixedUpdate() so it is stable.

Try to keep in mind, that if you want to add track and rewind functionality to the object that is regularly being changed in Update() method, it might result in slight synchronization problems, due to FixedUpdate() usually not catching with Update(). In rewind related stuff try to use FixedUpdate() instead of Update() whenever possible to completely avoid these issues. One simple way how to fix these problems if you need to use Update() method is to use **IsBeingRewinded** bool property from **RewindManager.cs**, and temporarily disabling Update() method while this property is true. Example of this simple solution is shown in **ParticleTimer.cs** (script is located in ExampleScenes/ScriptsForScene), where otherwise Update() and FixedUpdate() would fight against each other (although the problem in the script could be also solved just by using FixedUpdate()). Notice that also Slider Animator in prepared prefab uses FixedUpdate().



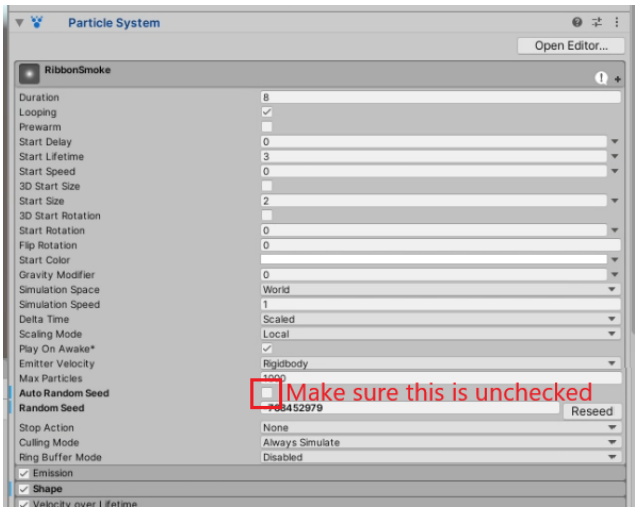
The reason why Time Rewinding is not implemented with the Update() method by default is, that it would add total unpredictability when rewinding by certain amount seconds. Also in my opinion, it is usually unecessary luxury to track values more often than in FixedUpdate() and it would also result in less overall performance.

Note that you can change intensity of FixedUpdate in Unity Time-settings preferences, but i advise you being carefull with that and have valid reasons before you start changing these values.

Particles System Rewinds

The only option that i have found to rewind particle systems is to use use Unity Particle.Simulate() method, which comes with huge performance hit if the particle system is running for long time especially in loop. Because of that i added limiter option for particle system tracking, that will reset the tracking after limit is hit. This will save a ton of performance in long particle systems. Although limiting particle system means that the rewind for long particle effects will not be so smooth everytime, it can be usually set up well enough to not see obvious jump transition. Short particle systems, dont need limiter and will rewind smoothly.

Important thing you must make sure you do, is to uncheck Auto Random Seed in ParticleSystem settings on every Particle System you want to rewind.



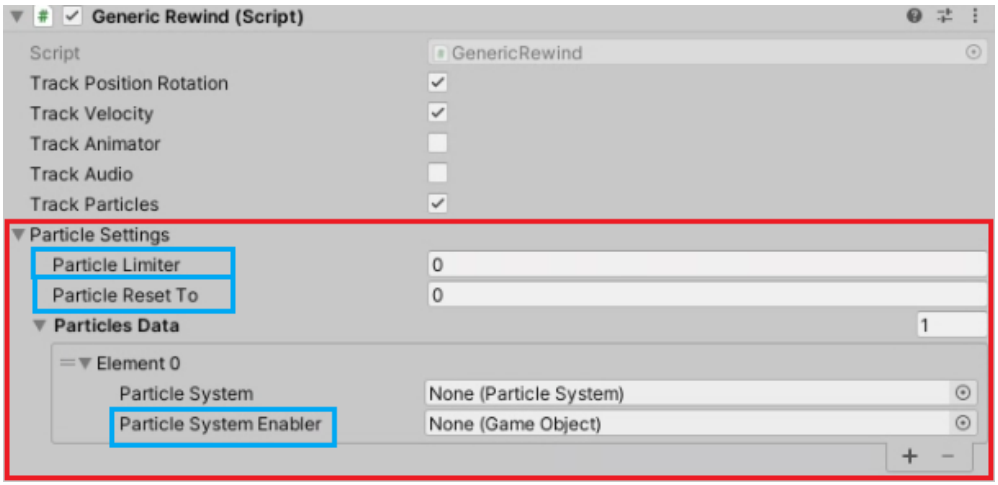
So the Particle.Simulate() is simulated with correct seed. You can set the Random Seed to whatever you like, just make sure you dont change it during Play time and during rewinds.

Before you start rewinding particles, you need to set up **Particle Settings.** When using **GenericRewind.cs** you only need to set it up in Unity Editor.

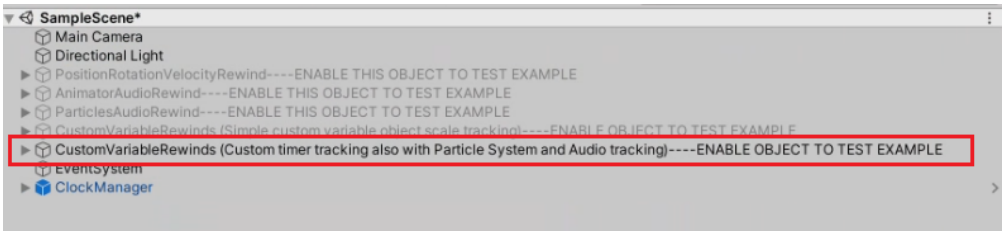
Particle Reset To shown on picture below is variable containing the time the Particle System will reset to after the Particle Limiter is hit. Try different values in this variable, to reach the smoothest possible transitions.

if your particle system is short enough (eg. 5-10 seconds)**,** set **Particle Limiter** to arbitraty number above your Particle System duration. If your Particle System is long, experiment with different values that will give you acceptable performance.

Particle System enabler shown also below is here to track particle active states, because Particle Systems are designed to be enabled or disabled during tracking. Particle System enabler should be GameObject of single Particle System that you want to track, or in case of many many Particle Systems and Sub-Particle Systems, it should be their Parent GameObject (also note that every sub Particle System must be included as one element of Particles Data).



Special example with enabling and disabling particle system is setup in last showcase in demoscene.



If you also want to setup your own custom tracker with Particle Tracking (similarly shown in **TimerRewind.cs** example), you must call **InitializeParticles()** in Start method of your own custom tracker. Initialization of particles is shown on example below.

```
public class TimerRewind : RewindAbstract
{
    CircularBuffer<float> trackedTime;              //For storing data, use this CircularBuffer class
    [SerializeField] ParticleTimer particleTimer;

    [SerializeField] ParticlesSetting particleSettings;

    Unity Message | 0 references
    private void Start()
    {
        InitializeParticles(particleSettings);      //When choosing to track particles in custom tracking script, you need to first initialize these particles in start method
    }
```

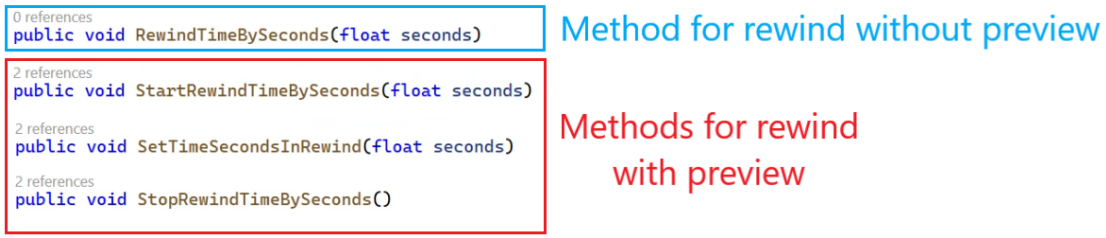
Rewinding time thru code

If you want to write your input rewind logic, you only need to call appropriate methods from **Rewind Manager**. I also suggest looking into **RewindByKeyPress**, which contains very easy to understand and simple implementation. **Manager** (more on that in last chapter of this document).

RewindBySlider.cs and **RewindByKeyPress.cs** are prepared for you, you can use it, customize it and visually enhance it in your game. However, if you want to rewind time by different input logic, you can write it yourself. The only thing

that is needed, is that you call appropriate methods from **RewindManager.cs** that are also documented. I also suggest looking into **RewindByKeyPress.cs**, which contains very easy to understand and simple implementation. **RewinManager.cs** provides two ways how to rewind time.

There are 4 main methods in **RewindManager.cs** that are important for you.



First way to rewind the time is with **RewindTimeBySeconds()** that is used for instant rewind, where you dont need rewind previews and you know you just want to rewind time by specified amount of seconds.

Second way of rewinding is with rewind previews (this way of doing rewinds is also shown in demo scene examples). Start rewinding time by calling **StartRewindTimeBySeconds()** method.

To update the preview when rewinding, use **SetTimeSecondsInRewing()** method to update the time of the preview you want to see.

After you are satisfied with currently shown preview and you want to return time to the shown preview, call method **StopRewindTimeBySeconds()**, which will stop the rewind and all stats shown in preview will be set to current objects, so game can continue from this point.