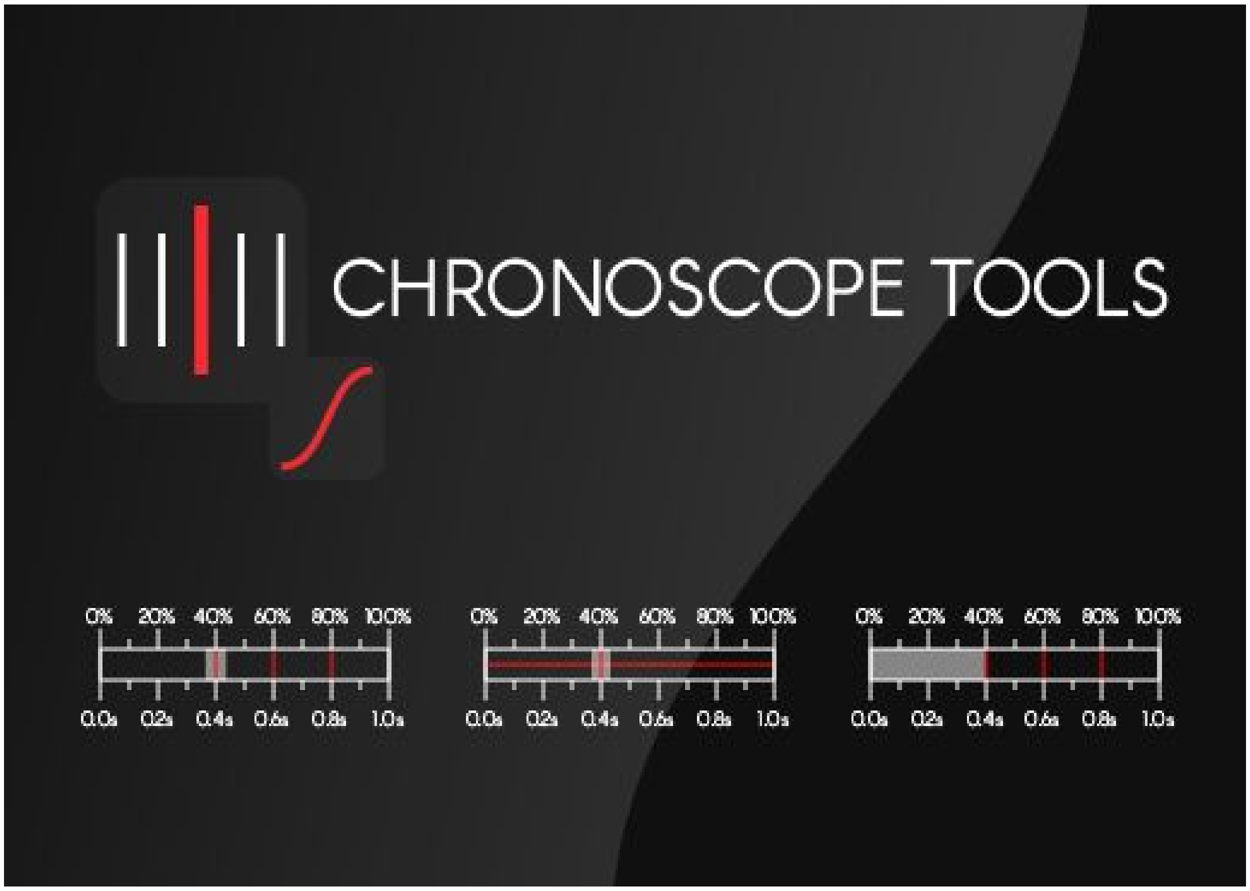


User Guide

Version 1.2 - 3rd July 2017

Contents

Profile

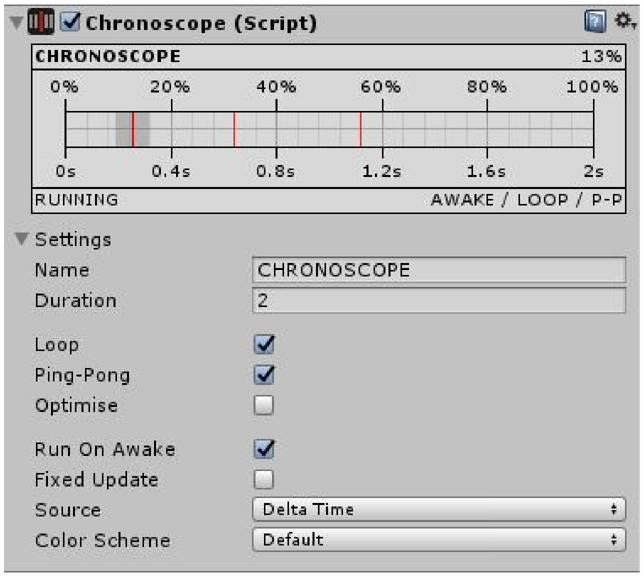
Apply

Overview

The Chronoscope toolset features three timers that operate in slightly different ways, all of the timers utilise coroutines to avoid cluttering the Unity Update.

Note: While every effort is made to keep this document up to date, the most recent information will always be available on the public wiki here.

Chronoscope



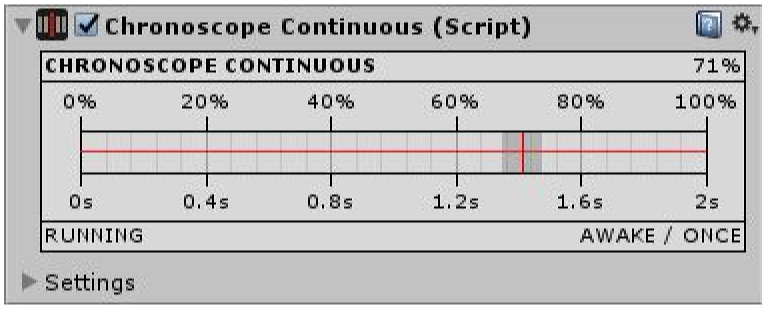
Settings

Name	CHRONOSCOPE
Duration	2
Loop	✓
Ping-Pong	✓
Optimise	☐
Run On Awake	✓
Fixed Update	☐
Source	Delta Time
Color Scheme	Default

This is the most versatile of the tools, it allows:

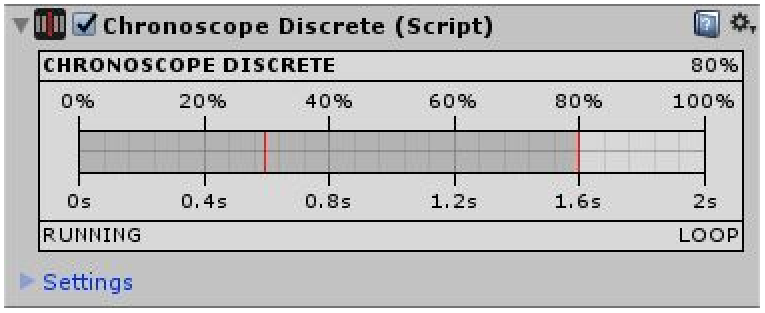
- Discrete Events
- Continuous Events
- Selectable Time Source
- Manual Pause & Resume

Chronoscope Continuous



This is very similar to the standard Chronoscope but without Discrete Events, this enables it to run a little more efficiently.

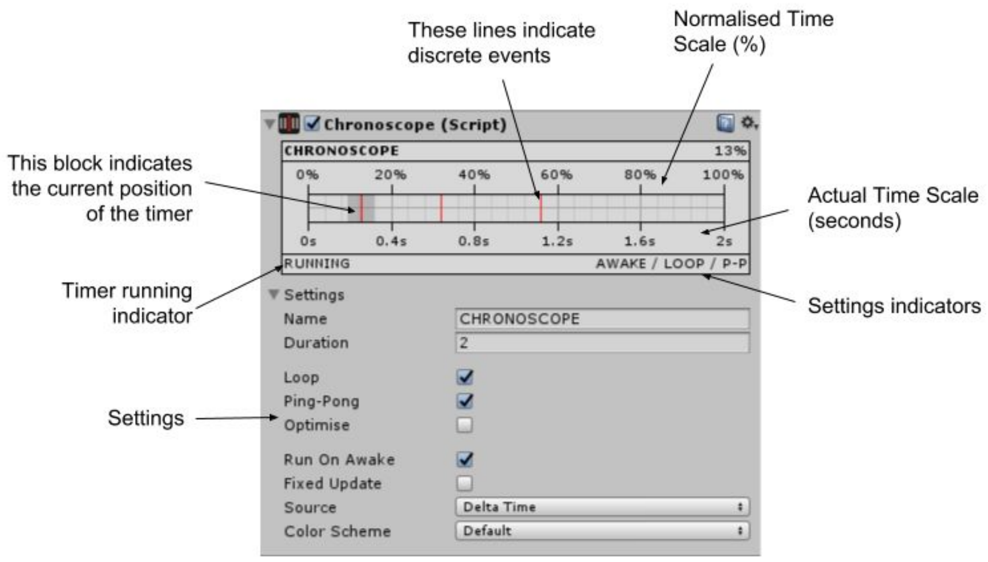
Chronoscope Discrete



This timer is optimised to allow only Discrete Events, and is perhaps the least flexible. It also has the limitation that it cannot be manually paused / resumed in code.

Chronoscope Timer

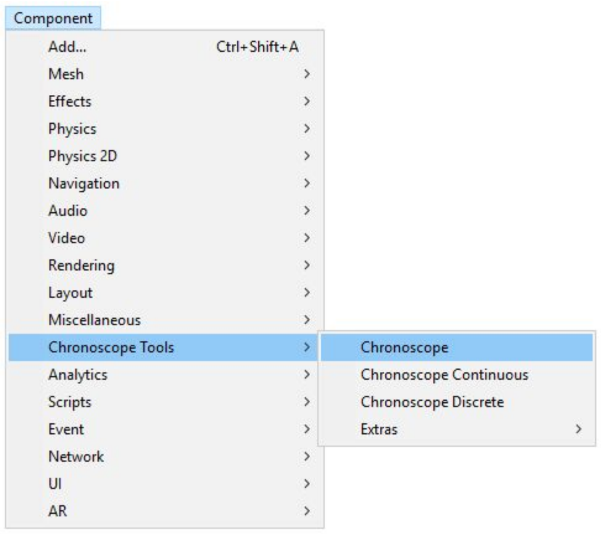
The Chronoscope timer is the most versatile of the tools, it provides everything that the ChronoscopeContinuous and ChronoscopeDiscrete timers offer but as a consequence doesn't run quite as efficiently.


These lines indicate discrete events
Normalised Time Scale (%)
This block indicates the current position of the timer
Actual Time Scale (seconds)
Timer running indicator
Settings indicators
Settings
Chronoscope (Script)
CHRONOSCOPE
13%
0% 20% 40% 60% 80% 100%
0s 0.4s 0.8s 1.2s 1.6s 2s
RUNNING
AWAKE / LOOP / P-P

Settings

Name	CHRONOSCOPE
Duration	2
Loop	✓
Ping-Pong	✓
Optimise	☐
Run On Awake	✓
Fixed Update	☐
Source	Delta Time
Color Scheme	Default

You can add this timer using the component menu..



Operation

This is a continually updated timer, it uses a coroutine that runs either every frame or after a fixed update frame (depending on the Fixed Update setting) and updates the timer value using the time source specified by the Source setting.

As it updates continually, this timer can be paused/resumed manually (i.e. by a method call rather than just by fiddling with the time scale as with ChronoscopeDiscrete).

Settings

The settings for the timer are described here.

Event Triggers

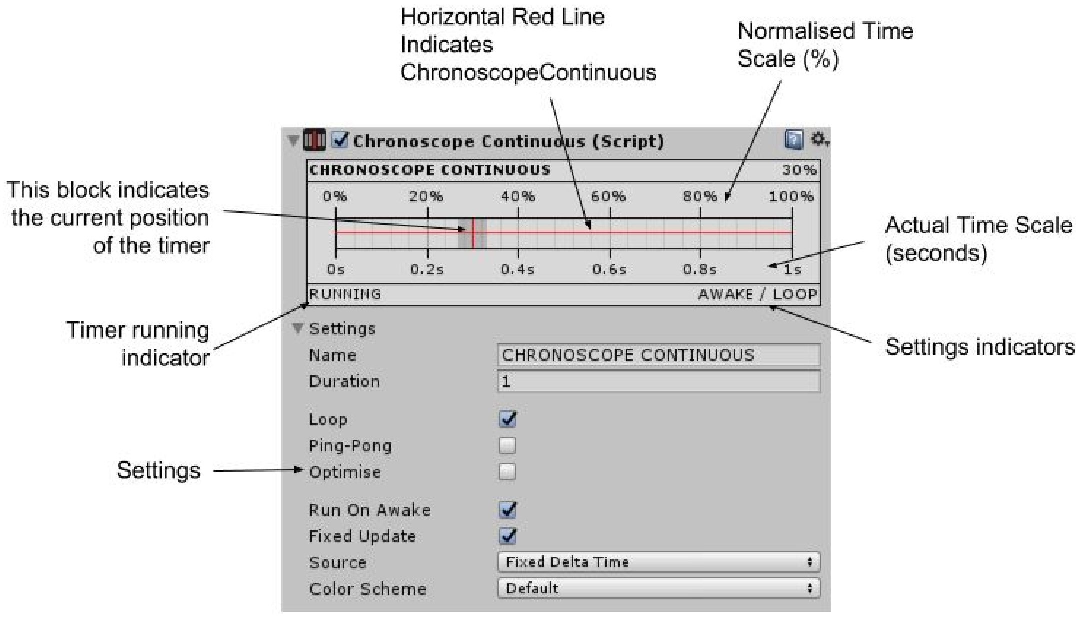
This timer can provide both Discrete and Continuous events, learn more here.

Control

The timer is controlled in code, using the `StartTimer`, `PauseTimer`, `ResetTimer` and `StopTimer` methods. See the API Reference for more info.

Chronoscope Continuous Timer

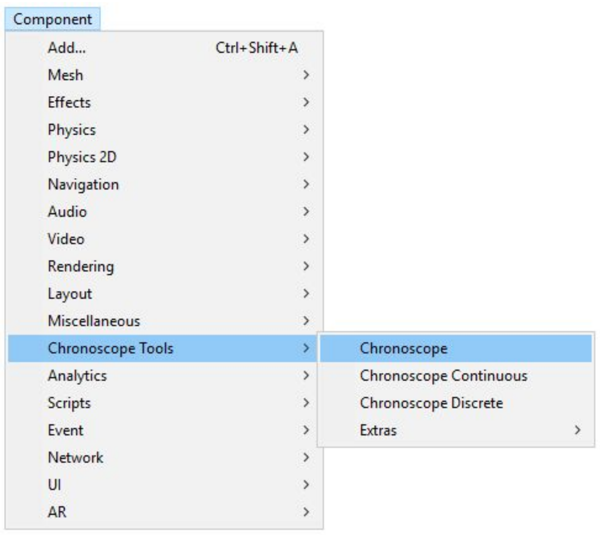
The Chronoscope timer is optimised for Continuous Events, but can still be manually paused / resumed in the same way as Chronoscope.


Horizontal Red Line Indicates ChronoscopeContinuous
Normalised Time Scale (%)
This block indicates the current position of the timer
Actual Time Scale (seconds)
Timer running indicator
Settings indicators
Settings
Chronoscope Continuous (Script)
CHRONOSCOPE CONTINUOUS
30%
0%
20%
40%
60%
80%
100%
0s
0.2s
0.4s
0.6s
0.8s
1s
RUNNING
AWAKE / LOOP

Settings

Name	CHRONOSCOPE CONTINUOUS
Duration	1
Loop	✓
Ping-Pong	☐
Optimise	☐
Run On Awake	✓
Fixed Update	✓
Source	Fixed Delta Time
Color Scheme	Default

You can add this timer using the component menu..



Operation

This is a continually updated timer, it uses a coroutine that runs either every frame or after a fixed update frame (depending on the Fixed Update setting) and updates the timer value using the time source specified by the Source setting.

As it updates continually, this timer can be paused/resumed manually (i.e. by a method call rather than just by fiddling with the time scale as with ChronoscopeDiscrete).

Settings

The settings for the timer are described here.

Event Triggers

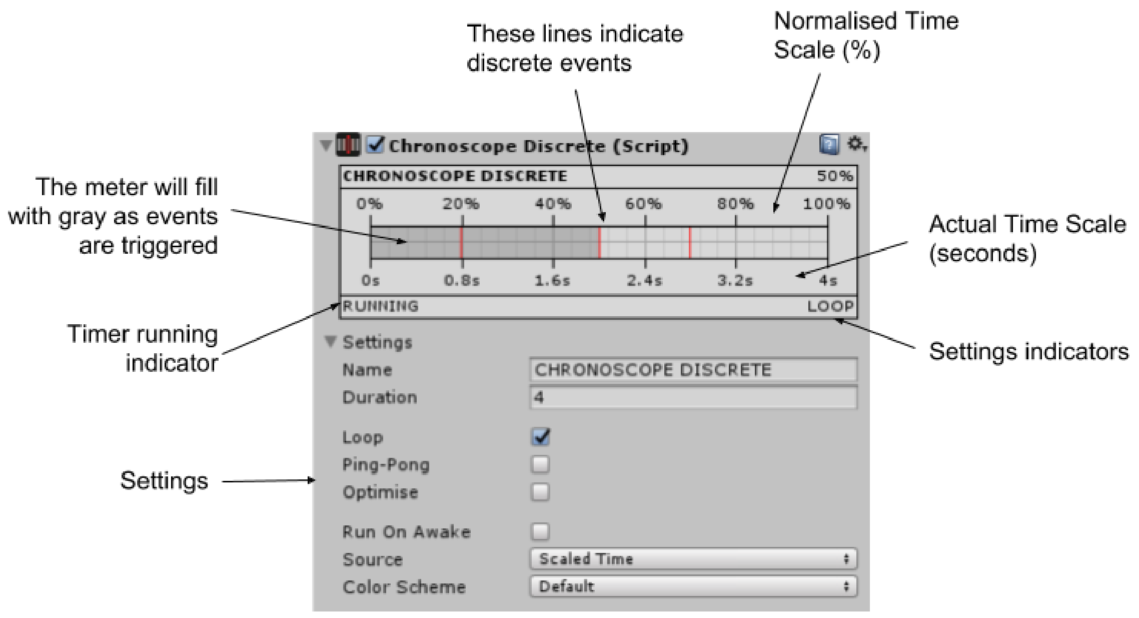
This timer can provide Continuous events, learn more here.

Control

The timer is controlled in code, using the `StartTimer`, `PauseTimer`, `ResetTimer` and `StopTimer` methods. See the API Reference for more info.

Chronoscope Discrete Timer

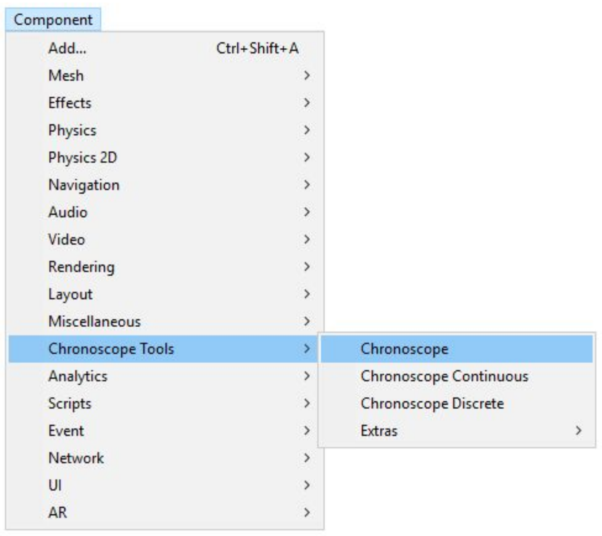
The ChronoscopeDiscrete timer is optimised for scheduling discrete events.


These lines indicate discrete events
Normalised Time Scale (%)
The meter will fill with gray as events are triggered
Actual Time Scale (seconds)
Timer running indicator
Settings indicators
Settings
Chronoscope Discrete (Script)
CHRONOSCOPE DISCRETE
50%
0%
20%
40%
60%
80%
100%
0s
0.8s
1.6s
2.4s
3.2s
4s
RUNNING
LOOP

Settings

Name	CHRONOSCOPE DISCRETE
Duration	4
Loop	✓
Ping-Pong	☐
Optimise	☐
Run On Awake	☐
Source	Scaled Time
Color Scheme	Default

You can add this timer using the component menu..



Operation

This is a *scheduled* timer, in that it does not update continually like the other two. When you start the timer, all the events a scheduled to be executed at their given times using coroutine waits. This means less code running perpetually.

The drawback of this is that the timer cannot be paused / resumed manually, and can only be paused by changing the Unity Time Scale to zero.. unless you set the timer Source to be `realTime` when even changing the Time Scale won't have any effect.

Settings

The settings for the timer are described here.

Event Triggers

This timer can provide Discrete events, learn more here.

Control

The timer is controlled in code, using the `StartTimer`, `ResetTimer` and `StopTimer` methods. See the API Reference for more info.

Timer Settings

The settings available in the inspector are described below.

Name

This is a descriptive name for the timer, useful for keeping things obvious and finding a particular timer when more than one is present on a GameObject.

Duration

The duration in seconds that the timer will run for.

Loop

When **Loop** is selected, the timer will automatically restart once it has finished.

Ping-Pong

When **Ping-Ping** is selected, once the timer has reached it's end it will count backwards to the start.

Optimise

This setting tells the timer to optimise the **Loop** and **Ping-Pong** settings when it is enabled, so that it doesn't waste time figuring out what to do every time it finishes. The downside to this is that any changes made in the editor at runtime won't have any effect.

Run On Awake

This setting determines whether the timer will start automatically when the GameObject is enabled, or if it needs to be started in code.

Fixed Update

If your timer is affecting any physics objects then this setting should be used. It will limit the timer to only updating after a fixed update frame.

Source

This determines which of Unity's time sources is used to update the value of the timer, and the available options differ depending on the timer type.

Chronoscope & Chronoscope Continuous

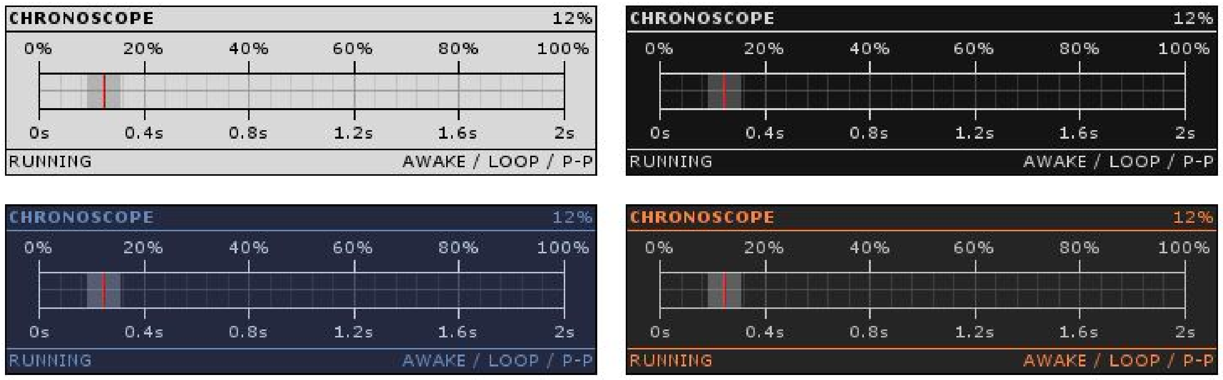
These have the most options available, refer to `TimerSourceEnum` in the API Reference.

Chronoscope Discrete

As this timer relies on scheduled coroutines rather than continuous updates it has fewer options available, refer to `WaitSourceEnum` in the API Reference.

Colour Scheme

Purely cosmetic, a number of schemes are available to pretty up your inspector if you don't like the default. If you wish to change the available schemes you should have a look at /Chronoscope/Misc/ChronoscopeColorPresets.cs in your Unity assets folder.



Timer Events

Continuous Events

Continuous events are available on the Chronoscope and ChronoscopeContinuous timers. These are events that are triggered every time the timer is updated.

A method can be hooked into the continuous events of a timer using the `AddContinuousListener` method, the method must accept a single float parameter that will be used to communicate the current normalised value of the timer.

Example

```
public Chronoscope myChronoscopeTimer;

void Start()
{
    myChronoscopeTimer.AddContinuousListener(ContinuousTimerListener);
    myChronoscopeTimer.Start(true);
}

// This method will be called by the timer every time it updates
void ContinuousTimerListener(float normalisedTimerValue)
{
    Debug.Log("Current Normalised Timer Value: ", normalisedTimerValue);
}
```

Discrete Events

Discrete events are available on the Chronoscope and ChronoscopeDiscrete timers. These are events that are triggered at a particular time.

A method can be hooked into the discrete events of a timer using the `AddDiscreteListener` method, the method must accept no parameters and a normalised time must be provided when adding the listener.

Example

```
public Chronoscope myChronoscopeTimer;

void Start()
{
    // Adds the 'DiscreteListener' method to the timer at 50%
    myChronoscopeTimer.AddDiscreteListener(0.5f, DiscreteTimerListener);
    myChronoscopeTimer.Start(true);
}

// This method will be called when the timer reaches 50%
void DiscreteTimerListener()
{
    Debug.Log("Discrete Event Called!");
}
```

If you need to convert between normalised and actual time for a given timer you can use the `ActualToNormalisedTime` and `NormalisedToActualTime` methods.

Special Events

Two special events are provided:

OnTimerStart()

This event is available on Chronoscope, ChronoscopeContinuous and ChronoscopeDiscrete. It is triggered when the timer is started.

OnTimerComplete()

This event is only available on Chronoscope and ChronoscopeContinuous, it is triggered when the timer reaches its end.

Example

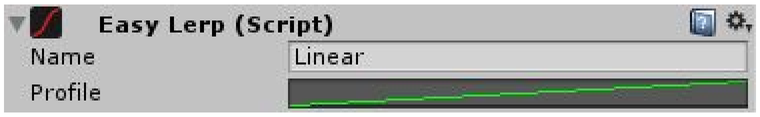
These are C# events that can have a DiscreteListener registered to them using the += syntax as shown below.

```
public Chronoscope myChronoscopeTimer;

void Start()
{
    myChronoscopeTimer.OnTimerStart += OnTimerStartListener;
    myChronoscopeTimer.OnTimerComplete += OnTimerCompleteListener;
}

void OnTimerStartListener()
{
    // Action to be performed on timer start
}

void OnTimerCompleteListener()
{
    // Action to be performed on timer complete
}
```

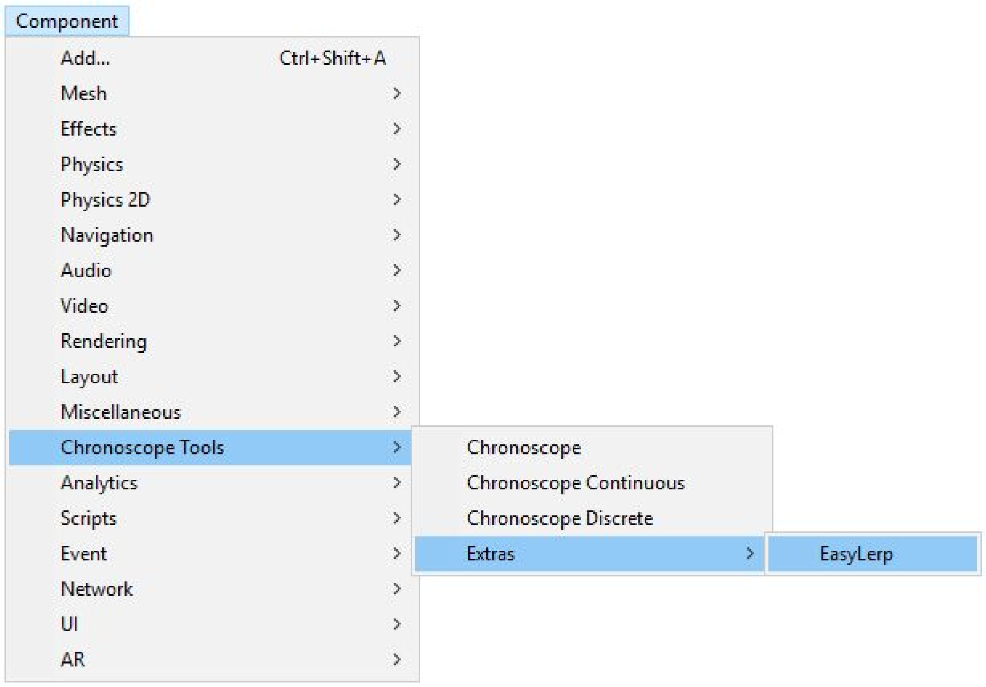
EasyLerp

EasyLerp uses a Unity `AnimationCurve` as a profile that can be used to lerp between two points, it also includes methods to do just that (provided you're lerping between floats, Vector2's or Vector3's).

There's really not much to this component, but it's so handy when used in conjunction with the timers that I wanted to include it.


Easy Lerp (Script)
Name: Linear
Profile

EasyLerp can be added from the component menu..



Operation

When combined with the normalised output of a Chronoscope timer, EasyLerp can make your life much easier. See the examples provided in the asset package (and described on the wiki) for ideas on how it might help you.

Settings

Name

This is descriptive name that can be used for distinguishing between multiple EasyLerp components.

Profile

The profile curve is edited in the inspector as normal, values at time > 1 or time < 0 will be ignored.

Apply

The 'Apply' method is more or less all there is to using this tool, see the API Reference for more information.