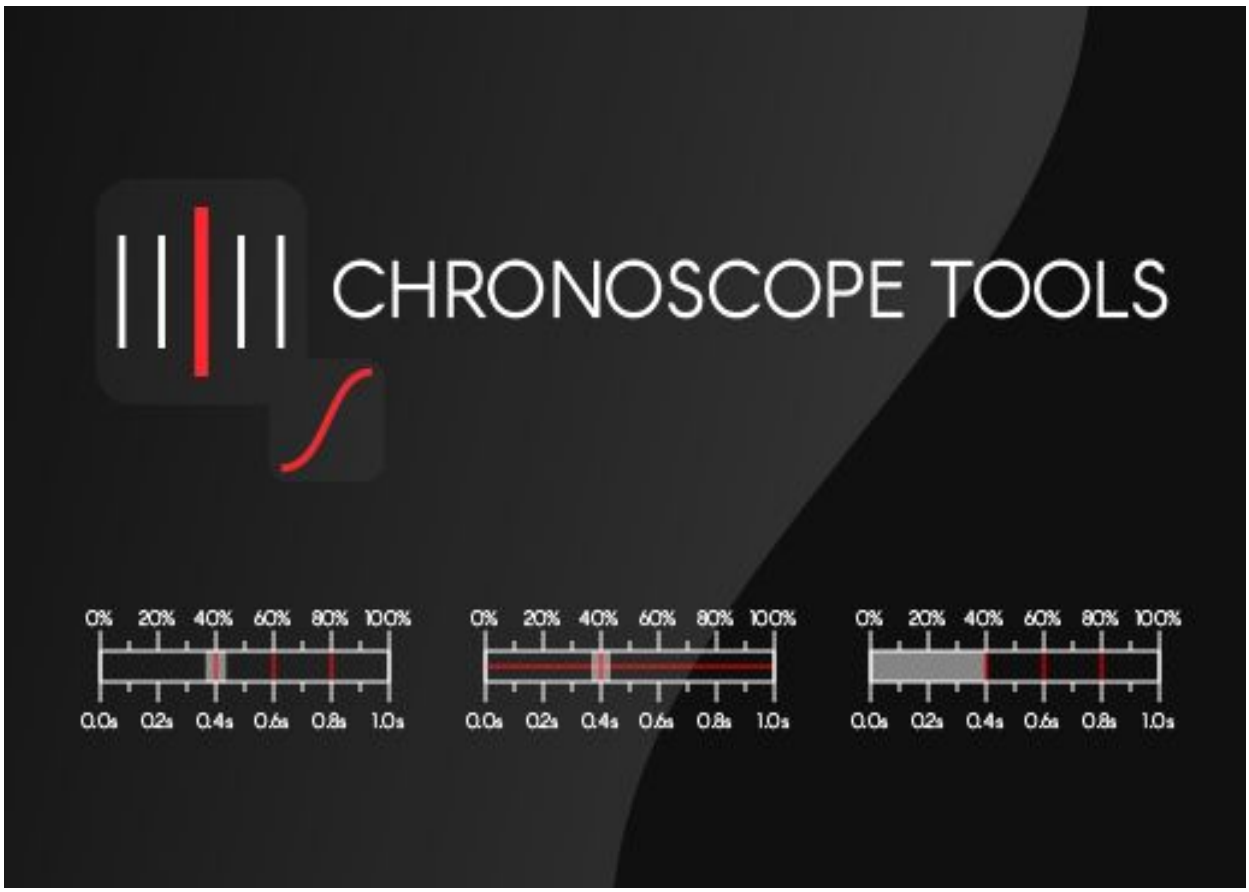


# API Reference

Version 1.2 - 3rd July 2017

# Contents

# Overview

This document describes the API used for the Chronoscope Tools.

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, in a much more easily navigable form!

# Namespace

To use and reference Chronoscope classes in your code you must add the following namespace to your script:

```
using ChronoscopeTools;
```

# Monobehaviour Classes

## Chronoscope

The Chronoscope class is a Monobehaviour component that provides continuous and discrete event triggers. More information about this component can be found here in the User Guide.

### Variables

| Name | Description |
|---|---|
| Name | Descriptive name of the timer |
| Running | Timer running flag |
| Loop | Loop setting |
| PingPong | Ping-Pong setting |
| RunOnAwake | Will the timer run automatically on start |
| Duration | The duration in seconds of the timer |
| Value | The current value of the timer in seconds |
| NormalisedValue | The normalised value of the timer |
| Source | The time source used to update the timer |
| FixedUpdate | Does the timer affect physics objects |
| TimerDirectionFlag | Current direction of the timer (see Ping-Pong) |

### Name

A descriptive name for the timer to help find it when there are multiple Chronoscope components on an object.

*Note: This property can only be edited in the inspector.*

```
public string Name { get; }
```

**ACCESS:** Read Only

---

### Running

Indicates whether or not the timer is currently running (`true` if running).

*Note: This property can only be edited in the inspector.*

```
public bool Running { get; }
```

**ACCESS:** Read Only

---

### Loop

Determines whether or not the time loops back to the start when it finishes.

*Note: This property can only be edited in the inspector.*

```
public bool Loop { get; }
```

**ACCESS:** Read Only

---

### PingPong

When true, upon reaches reaching it's end point the timer will reverse and count back to the start, before looping or stopping depending on the `Loop` setting.

*Note: This property can only be edited in the inspector.*

```
public bool PingPong { get; }
```

**ACCESS:** Read Only

---

RunOnAwake

When true, the timer will start when the object is enabled.

*Note: This property can only be edited in the inspector.*

```
public bool RunOnAwake { get; }
```

**ACCESS:** Read Only

---

Duration

The duration in seconds that the timer will run for.

```
public float Duration { get; set; }
```

**ACCESS:** Read / Write

---

Value

The current value (in seconds) of the timer.

```
public float Value { get; }
```

**ACCESS:** Read Only

---

NormalisedValue

The current normalised value of the timer, i.e. a percentage expressed as a value between 0 and 1.

```
public float NormalisedValue { get; }
```

**ACCESS:** Read Only

---

Source

Selects which time source is used to update the timer value.

*Note: This property is optimized on initialization of the timer, so any changes made to it during runtime will have no effect.*

```
public TimerSourceEnum Source { get; set; }
```

**ACCESS:** Read / Write

---

FixedUpdate

Determines whether this timer should only be updated after a fixed update frame, set this to true if the timer affects any physics objects.

```
public bool FixedUpdate { get; set; }
```

**ACCESS:** Read / Write

---

TimerDirectionFlag

Indicates the current direction of the timer (up or down).

```
public TimerDirection TimerDirectionFlag { get; }
```

**ACCESS:** Read Only

---

## Public Methods

| Name | Description |
|---|---|
| **StartTimer** | Starts/Resumes the timer |
| **PauseTimer** | Pauses the timer (without resetting it) |
| **StopTimer** | Stops and resets the timer |
| **ResetTimer** | Resets the timer |
| **AddDiscreteListener** | Adds a listener event at a specified time |
| **RemoveDiscreteListener** | Removes a listener event at the specified time |
| **RemoveAllDiscreteListeners** | Removes all discrete triggers listeners |
| **AddContinuousListener** | Adds a continuous listeners |
| **RemoveContinuousListener** | Removes a continuous listener |
| **RemoveAllContinuousListeners** | Removes all continuous listeners |

| | |
|---|---|
| **ActualToNormalisedTime** | Helper to convert time in seconds to normalised time |
| **NormalisedToActualTime** | Helper to convert normalised time to seconds |

### StartTimer

Starts / Resumes the timer with an option reset.

```
public void StartTimer(bool reset)
```

| Parameter | Description |
|---|---|
| **reset** | If true the timer will reset before starting |

---

### PauseTimer

Pauses the timer without resetting it.

```
public void PauseTimer()
```

---

### StopTimer

Stops and resets the timer

```
public void StopTimer()
```

---

### ResetTimer

Resets the timer

```
public void ResetTimer()
```

---

### AddDiscreteListener

Adds a listener method at the specified normalized event time.

```
public void AddDiscreteListener(float normalisedEventTime, DiscreteListener listener)
```

| Parameter | Description |
|---|---|

| | |
|---|---|
| **normalisedEventTime** | The normalised (between 0 and 1) time that the listener should be triggered at |
| **listener** | The listener method to be triggered |

---

RemoveDiscreteListener

Removes a listener method at the specified normalised event time.

```
public bool RemoveDiscreteListener(float normalisedEventTime, DiscreteListener listener)
```

| Parameter | Description |
|---|---|
| **normalisedEventTime** | The normalized (between 0 and 1) time that the listener is registered at |
| **listener** | The listener method to be removed |

***Returns*** true if the listener was removed, or false if it was not found.

---

RemoveAllDiscreteListeners

Removes all discrete listeners from the timer.

```
public void RemoveAllDiscreteListeners()
```

---

AddContinuousListener

Adds a listener method that will be called every timer the timer updates.

```
public void AddContinuousListener(ContinuousListener listener)
```

| Parameter | Description |
|---|---|
| **listener** | The listener method to be called by the timer |

---

### RemoveContinuousListener

Removes a continuous listener method from the timer.

```
public bool RemoveContinuousListener(ContinuousListener listener)
```

| Parameter | Description |
| --- | --- |
| **listener** | The listener method to be removed |

***Returns*** true if the listener was removed, or false if it was not found.

---

### RemoveAllContinuousListeners

Removes all continuous listeners from the timer.

```
public void RemoveAllContinuousListeners()
```

---

### RemoveAllContinuousListeners

Removes all continuous listeners from the timer.

```
public void RemoveAllContinuousListeners()
```

---

### ActualToNormalisedTime

Converts a timer value in seconds to a normalised timer value (i.e. a value between 0 and 1).

```
public float ActualToNormalisedTime(float actualTimeValue)
```

| Parameter | Description |
| --- | --- |
| **actualTimeValue** | The time in seconds to convert to a normalised value |

***Returns*** the normalised value of the provided time.

---

NormalisedToActualTime

Converts a normalised timer value to a time in seconds.

```
public float NormalisedToActualTime(float normalisedTimerValue)
```

| Parameter | Description |
| --- | --- |
| **normalisedTimerValue** | The normalised value to convert |

***Returns*** the time in seconds of the provided normalised time.

---

## Events

| Name | Description |
| --- | --- |
| **OnTimerStart** | Triggered when the timer is started |
| **OnTimerComplete** | Triggered when the timer reaches its end |

OnTimerStart

Triggered when the timer is started.

```
public event DiscreteListener OnTimerStart
```

---

OnTimerComplete

Triggered when the timer reaches its end

```
public event DiscreteListener OnTimerComplete
```

## ChronoscopeContinuous

The ChronoscopeContinuous class is a Monobehaviour component that provides continuous event triggers. More information about this component can be found in the User Guide.

### Variables

| Name | Description |
|---|---|
| **Name** | Descriptive name of the timer |
| **Running** | Timer running flag |
| **Loop** | Loop setting |
| **PingPong** | Ping-Pong setting |
| **RunOnAwake** | Will the timer run automatically on start |
| **Duration** | The duration in seconds of the timer |
| **Value** | The current value of the timer in seconds |
| **NormalisedValue** | The normalised value of the timer |
| **Source** | The time source used to update the timer |
| **FixedUpdate** | Does the timer affect physics objects |
| **TimerDirectionFlag** | Current direction of the timer (see Ping-Pong) |

Name

A descriptive name for the timer to help find it when there are multiple Chronoscope components on an object.

*Note: This property can only be edited in the inspector.*

```
public string Name { get; }
```

**ACCESS:** Read Only

---

Running

Indicates whether or not the timer is currently running (`true` if running).

*Note: This property can only be edited in the inspector.*

```
public bool Running { get; }
```

**ACCESS:** Read Only

---

Loop

Determines whether or not the time loops back to the start when it finishes.

*Note: This property can only be edited in the inspector.*

```
public bool Loop { get; }
```

**ACCESS:** Read Only

---

PingPong

When true, upon reaches reaching it's end point the timer will reverse and count back to the start, before looping or stopping depending on the `Loop` setting.

*Note: This property can only be edited in the inspector.*

```
public bool PingPong { get; }
```

**ACCESS:** Read Only

---

RunOnAwake

When true, the timer will start when the object is enabled.

*Note: This property can only be edited in the inspector.*

```
public bool RunOnAwake { get; }
```

**ACCESS:** Read Only

---

Duration

The duration in seconds that the timer will run for.

```
public float Duration { get; set; }
```

**ACCESS:** Read / Write

---

Value

The current value (in seconds) of the timer.

```
public float Value { get; }
```

**ACCESS:** Read Only

---

NormalisedValue

The current normalised value of the timer, i.e. a percentage expressed as a value between 0 and 1.

```
public float NormalisedValue { get; }
```

**ACCESS:** Read Only

---

Source

Selects which time source is used to update the timer value.

*Note: This property is optimized on initialization of the timer, so any changes made to it during runtime will have no effect.*

```
public TimerSourceEnum Source { get; set; }
```

**ACCESS:** Read / Write

---

FixedUpdate

Determines whether this timer should only be updated after a fixed update frame, set this to true if the timer affects any physics objects.

```
public bool FixedUpdate { get; set; }
```

**ACCESS:** Read / Write

---

TimerDirectionFlag

Indicates the current direction of the timer (up or down).

```
public TimerDirection TimerDirectionFlag { get; }
```

**ACCESS:** Read Only

---

## Public Methods

| Name | Description |
|---|---|
| **StartTimer** | Starts/Resumes the timer |
| **PauseTimer** | Pauses the timer (without resetting it) |
| **StopTimer** | Stops and resets the timer |
| **ResetTimer** | Resets the timer |
| **AddContinuousListener** | Adds a continuous listeners |
| **RemoveContinuousListener** | Removes a continuous listener |
| **RemoveAllContinuousListeners** | Removes all continuous listeners |
| **ActualToNormalisedTime** | Helper to convert time in seconds to normalised time |
| **NormalisedToActualTime** | Helper to convert normalised time to seconds |

StartTimer

Starts / Resumes the timer with an option reset.

```
public void StartTimer(bool reset)
```

| Parameter | Description |
|---|---|
| **reset** | If true the timer will reset before starting |

---

PauseTimer

Pauses the timer without resetting it.

```
public void PauseTimer()
```

---

StopTimer

Stops and resets the timer

```
public void StopTimer()
```

---

ResetTimer

Resets the timer

```
public void ResetTimer()
```

---

AddContinuousListener

Adds a listener method that will be called every timer the timer updates.

```
public void AddContinuousListener(ContinuousListener listener)
```

| Parameter | Description |
| --- | --- |
| **listener** | The listener method to be called by the timer |

---

RemoveContinuousListener

Removes a continuous listener method from the timer.

```
public bool RemoveContinuousListener(ContinuousListener listener)
```

| Parameter | Description |
| --- | --- |
| **listener** | The listener method to be removed |

***Returns*** true if the listener was removed, or false if it was not found.

---

RemoveAllContinuousListeners

Removes all continuous listeners from the timer.

```
public void RemoveAllContinuousListeners()
```

---

RemoveAllContinuousListeners

Removes all continuous listeners from the timer.

```
public void RemoveAllContinuousListeners()
```

---

ActualToNormalisedTime

Converts a timer value in seconds to a normalised timer value (i.e. a value between 0 and 1).

```
public float ActualToNormalisedTime(float actualTimeValue)
```

| Parameter | Description |
| --- | --- |
| **actualTimeValue** | The time in seconds to convert to a normalised value |

***Returns*** the normalised value of the provided time.

---

NormalisedToActualTime

Converts a normalised timer value to a time in seconds.

```
public float NormalisedToActualTime(float normalisedTimerValue)
```

| Parameter | Description |
| --- | --- |
| **normalisedTimerValue** | The normalised value to convert |

***Returns*** the time in seconds of the provided normalised time.

---

## Events

| Name | Description |
| --- | --- |
| **OnTimerStart** | Triggered when the timer is started |
| **OnTimerComplete** | Triggered when the timer reaches its end |

### OnTimerStart

Triggered when the timer is started.

```
public event DiscreteListener OnTimerStart
```

---

### OnTimerComplete

Triggered when the timer reaches its end

```
public event DiscreteListener OnTimerComplete
```

## ChronoscopeDiscrete

The Chronoscope class is a Monobehaviour component that provides discrete event triggers. More information about this component can be found in the User Guide.

### Variables

| Name | Description |
|---|---|
| Name | Descriptive name of the timer |
| Running | Timer running flag |
| Loop | Loop setting |
| PingPong | Ping-Pong setting |
| RunOnAwake | Will the timer run automatically on start |
| Duration | The duration in seconds of the timer |
| Value | The current value of the timer in seconds |
| LastNormalisedEventTime | The normalised time of the last triggered event |
| Source | The time source used to update the timer |
| FixedUpdate | Does the timer affect physics objects |
| TimerDirectionFlag | Current direction of the timer (see Ping-Pong) |

Name

A descriptive name for the timer to help find it when there are multiple Chronoscope components on an object.

*Note: This property can only be edited in the inspector.*

```
public string Name { get; }
```

**ACCESS:** Read Only

---

Running

Indicates whether or not the timer is currently running (`true` if running).

*Note: This property can only be edited in the inspector.*

```
public bool Running { get; }
```

**ACCESS:** Read Only

---

Loop

Determines whether or not the time loops back to the start when it finishes.

*Note: This property can only be edited in the inspector.*

```
public bool Loop { get; }
```

**ACCESS:** Read Only

---

PingPong

When true, upon reaches reaching it's end point the timer will reverse and count back to the start, before looping or stopping depending on the `Loop` setting.

*Note: This property can only be edited in the inspector.*

```
public bool PingPong { get; }
```

**ACCESS:** Read Only

---

RunOnAwake

When true, the timer will start when the object is enabled.

*Note: This property can only be edited in the inspector.*

```
public bool RunOnAwake { get; }
```

**ACCESS:** Read Only

---

Duration

The duration in seconds that the timer will run for.

```
public float Duration { get; set; }
```

**ACCESS:** Read / Write

---

Value

The current value (in seconds) of the timer.

```
public float Value { get; }
```

**ACCESS:** Read Only

---

LastNormalisedEventTime

The normalised time of the last event that was triggered.

```
public float LastNormalisedEventTime { get; }
```

**ACCESS:** Read Only

---

Source

Selects which time source is used to update the timer value.

*Note: This property is optimized on initialization of the timer, so any changes made to it during runtime will have no effect.*

```
public WaitSourceEnum Source { get; set; }
```

**ACCESS:** Read / Write

---

FixedUpdate

Determines whether this timer should only be updated after a fixed update frame, set this to true if the timer affects any physics objects.

```
public bool FixedUpdate { get; set; }
```

**ACCESS:** Read / Write

---

TimerDirectionFlag

Indicates the current direction of the timer (up or down).

```
public TimerDirection TimerDirectionFlag { get; }
```

**ACCESS:** Read Only

---

## Public Methods

| Name | Description |
|---|---|
| **StartTimer** | Starts the timer |
| **StopTimer** | Stops and resets the timer |
| **ResetTimer** | Resets the timer |
| **AddDiscreteListener** | Adds a listener event at a specified time |
| **RemoveDiscreteListener** | Removes a listener event at the specified time |
| **RemoveAllDiscreteListeners** | Removes all discrete triggers listeners |
| **ActualToNormalisedTime** | Helper to convert time in seconds to normalised time |
| **NormalisedToActualTime** | Helper to convert normalised time to seconds |

StartTimer

Starts the timer.

```
public void StartTimer()
```

---

StopTimer

Stops and resets the timer

```
public void StopTimer()
```

---

ResetTimer

Resets the timer

```
public void ResetTimer()
```

---

AddDiscreteListener

Adds a listener method at the specified normalized event time.

```
public void AddDiscreteListener(float normalisedEventTime, DiscreteListener listener)
```

| Parameter | Description |
| --- | --- |
| **normalisedEventTime** | The normalised (between 0 and 1) time that the listener should be triggered at |
| **listener** | The listener method to be triggered |

---

RemoveDiscreteListener

Removes a listener method at the specified normalised event time.

```
public bool RemoveDiscreteListener(float normalisedEventTime, DiscreteListener listener)
```

| Parameter | Description |
| --- | --- |
| **normalisedEventTime** | The normalized (between 0 and 1) time that the listener is registered at |
| **listener** | The listener method to be removed |

***Returns*** true if the listener was removed, or false if it was not found.

---

### RemoveAllDiscreteListeners

Removes all discrete listeners from the timer.

```
public void RemoveAllDiscreteListeners()
```

---

### ActualToNormalisedTime

Converts a timer value in seconds to a normalised timer value (i.e. a value between 0 and 1).

```
public float ActualToNormalisedTime(float actualTimeValue)
```

| Parameter | Description |
| --- | --- |
| **actualTimeValue** | The time in seconds to convert to a normalised value |

***Returns*** the normalised value of the provided time.

---

### NormalisedToActualTime

Converts a normalised timer value to a time in seconds.

```
public float NormalisedToActualTime(float normalisedTimerValue)
```

| Parameter | Description |
| --- | --- |
| **normalisedTimerValue** | The normalised value to convert |

***Returns*** the time in seconds of the provided normalised time.

---

## Events

| Name | Description |
| --- | --- |
| **OnTimerStart** | Triggered when the timer is started |

OnTimerStart

Triggered when the timer is started.

```
public event DiscreteListener OnTimerStart
```

## EasyLerp

The EasyLerp class is a Monobehaviour component that provides an interpolation profile. For more information see the User Guide.

### Variables

Name

A descriptive name for the profile, useful when there are multiple EasyLerp components on a single object.

*Note: This property can only be edited in the inspector.*

```
public string Name { get; }
```

**ACCESS:** Read Only

### Public Methods

Apply

Overloaded method that will either return the interpolated value from the provided alpha or provide a point interpolated between two points using the profile.

```
public float Apply(float alpha);
public float Apply(float start, float end, float alpha);
public Vector2 Apply(Vector2 start, Vector2 end, float alpha);
public Vector3 Apply(Vector3 start, Vector3 end, float alpha);
```

*Overloads*

```
public float Apply(float alpha);
```

| Parameter | Description |
| --- | --- |
| **alpha** | The normalized input to evaluate |

***Returns*** the evaluated value.

```
public float Apply(float start, float end, float alpha);
public Vector2 Apply(Vector2 start, Vector2 end, float alpha);
public Vector3 Apply(Vector3 start, Vector3 end, float alpha);
```

| Parameter | Description |
| --- | --- |
| **start** | The start value to lerp from |
| **end** | The end value to lerp to |
| **alpha** | The normalized input to evaluate |

***Returns*** the point between the start and end given by the evaluated alpha.

# Enums

## TimerSourceEnum

Used for setting the timer source of continuous update timers.

```
public enum TimerSourceEnum
{
    deltaTime,
    unscaledDeltaTime,
    fixedDeltaTime,
    fixedUnscaledDeltaTime,
    smoothDeltaTime
}
```

*Refer to the `Time` class in the Unity3D scripting reference for detailed information on these time sources.*

## WaitSourceEnum

Used for setting the timer source of discrete update timers.

```
public enum WaitSourceEnum
{
    scaledTime,
    realTime
}
```

## TimerDirection

Used to indicate the current direction of timers.

```
public enum TimerDirection
{
    up,
    down
}
```

# Delegates

## ContinuousListener

A delegate used for adding continuous event listeners to timers.

```
public delegate void ContinuousListener(float normalisedTimerValue);
```

| Parameter | Description |
| --- | --- |
| **normalisedTimerValue** | The current normalised time provided by the timer |

## DiscreteListener

A delegate used for adding discrete event listeners to timers.

```
public delegate void DiscreteListener();
```