EASY TIMER

Easy Timer provides a simple but extremely useful class for quickly getting timer values in a variety of interpolation methods. The basic steps include:

1.] Create a reference for your Timer
2.] Instantiate the Timer when you are ready to begin counting
3.] Get one of many Timer properties to drive all your timing needs

JavaScript (UnityScript)

```
#pragma strict

public class TestTimer extends MonoBehaviour
{
        // Step 1
        var timer : Timer;

        function Start ()
        {
                // Step 2
                timer  = new Timer( 4 );
        }

        function Update ()
        {
                // Step 3
                Debug.Log ( timer.time );
        }
}
```

C#

```
using UnityEngine;
using System.Collections;

public class TestTimer : MonoBehaviour
{
        // Step 1
        Timer timer;

        void Start ()
        {
                // Step 2
                timer = new Timer(4);
        }


        void Update ()
        {
                // Step 3
                Debug.Log ( timer.time );
        }
}
```

! NOTE	To use Easy Timer with JavaScript (UnityScript), make sure the **Timer** script remains in the **Standard Assets** folder.

Constructors:

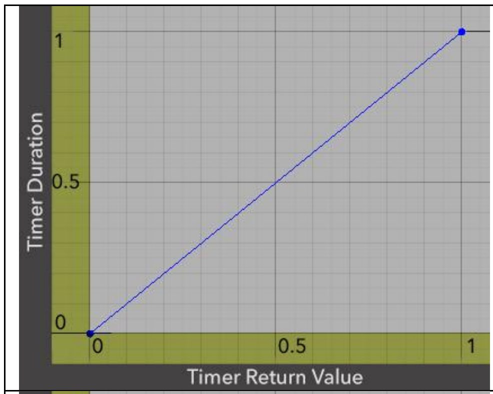
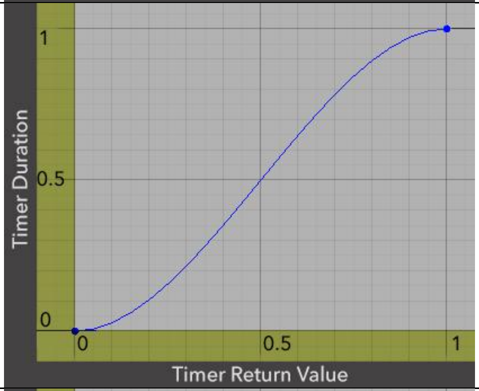
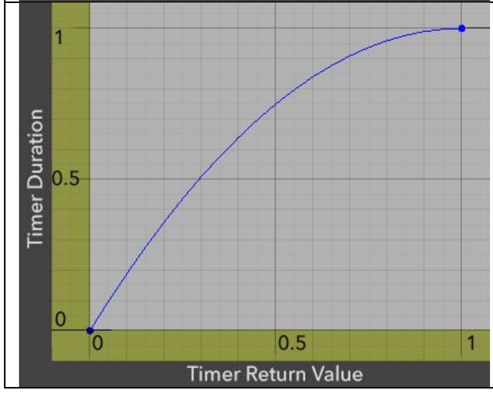
Timer ( ) : **Timer**
Returns a new instance of Timer, with default duration of 1 second.

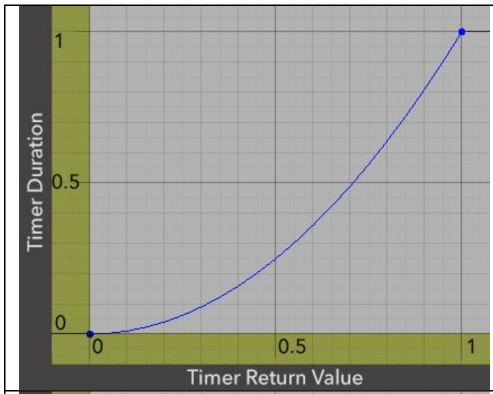
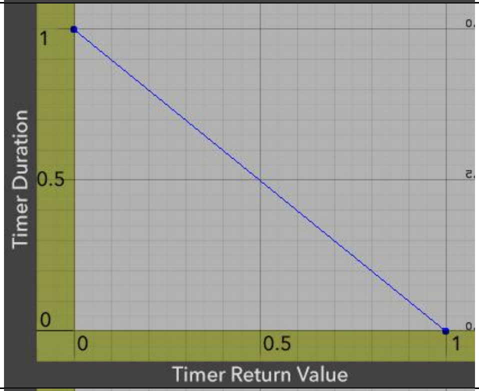
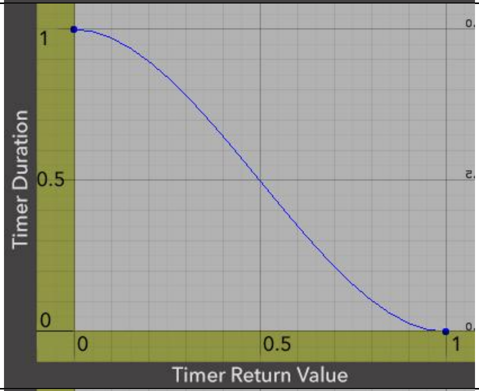
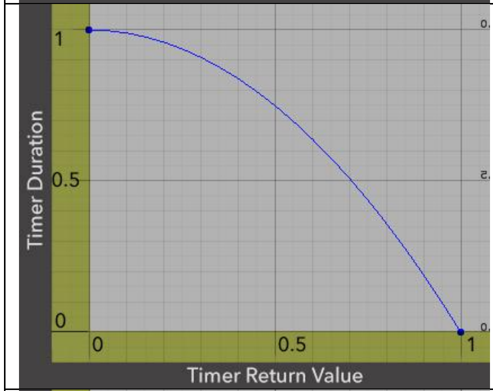
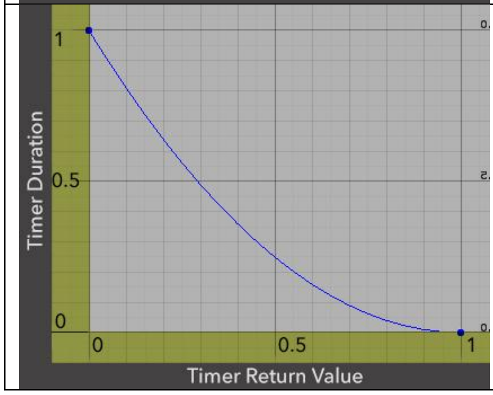
Timer (**float** *duration*) : **Timer**
Returns a new instance of Timer, with timer duration specified by *duration*.

Timer (**float** *duration*, **float** *delay*) : **Timer**
Returns a new instance of Timer, which will return **0** for *delay* seconds, and then will count down for *duration* seconds.

! NOTE	Do not instantiate a Timer from a field initializer. The internal call to **UnityEngine.Time.time** will cause an exception.

Properties

	**time** : float { get; } The normalized (0 to 1) return of the current timer position, evaluated over a linear interpolation.
	**timeSmooth** : float { get; } The normalized (0 to 1) return of the current timer position, Bezier interpolation with slow in and slow out.
	**timeFastIn**: float { get; } The normalized (0 to 1) return of the current timer position, Bezier interpolation with fast in and slow out.

	**timeFastOut**: float { get; } The normalized (0 to 1) return of the current timer position, Bezier interpolation with slow in and fast out.
	**timeInversed**: float { get; } The inversed normalized (0 to 1) return of the current timer position (1 – **time**), evaluated over a linear interpolation.
	**timeSmoothInversed**: float { get; } The inversed normalized (0 to 1) return of the current timer position (1 – **timeSmooth**), Bezier interpolation with slow in and slow out.
	**timeFastInInversed**: float { get; } The inversed normalized (0 to 1) return of the current timer position (1 – **timeFastIn**), Bezier interpolation with slow in and fast out.
	**timeFastOutInversed**: float { get; } The inversed normalized (0 to 1) return of the current timer position (1 – **timeFastOut**), Bezier interpolation with fast in and slow out.

timeUnClamped: float { get; }

The normalized (0 to 1) return of the current timer position allowed to extend beyond 1. (see example below)

timeTotal: float { get; }

The non-normalized and unclamped return of actual seconds for the current timer position. (see example below)

Example:

Constructor	After Seconds	timer.**time**	timer. **timeUnClamped**	timer.**timeTotal**
timer = Timer(2);	0	0.0	0.0	0.0
timer = Timer(2);	1	0.5	0.5	1.0
timer = Timer(2);	2	1.0	1.0	2.0
timer = Timer(2);	3	1.0	1.5	3.0
timer = Timer(2);	4	1.0	2.0	4.0