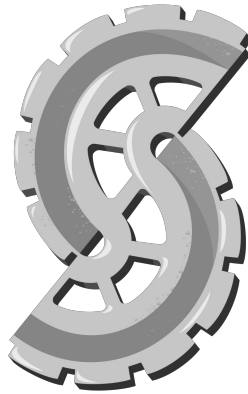


STR8TWEEN



documentation

Introduction

What is Str8Tween ? To answer quickly it's a tween engine.

But if you don't know about tweens and tween engines yet, here are some definitions.

What is a tween ?

The word "tween" comes from "between" and can be used as a noun or a verb.

In computer graphics, animation or cinematography it's the action of generating intermediates frames in an animated sequence to obtain smooth movements.

In mathematics it's a method of constructing new data points within the range of a set of known data points. An ease can be applied to a tween to modify the rate of change of the calculated data points over time.

In Str8Tween a tween is what holds the calculations of new values and apply them to the target.

To customise tweens I implemented [Robert Penner's](#) easing equations.

See easings.net to visualize each easing function.

What are tween engines ?

According to the previous definition a tween engine can be defined as a piece of software that performs tweening.

It provides methods to create tweens and manage them along their lifecycles.

Setup

Follow theses steps to get Str8Tween, to include it in your projects and to use it.

Installation

First you need to download the zip file, dll or the unity package.

To do so, go to the [download page](#).

Once you have downloaded one of the files refer to the appropriate paragraph below.

Zip or dll

If you downloaded the zip file, unzip it and add the str8tween.dll file to your "Assets" folder.

For more clarity you can add the dll to a "Plugins" folder as follow or any other custom path :

"Assets/Plugins/str8tween.dll".

And you're good to go.

Unity package

Unity packages need to be imported directly using the Unity Editor.

To import the package, go to Assets/Import Package/Custom Package in the editor menu.

Then select the path to the .unitypackage file and click on "open".

You'll have the following files added to your project at Assets/Plugins/Str8Tween/ :

- Str8Tween.dll
- README.md
- LICENSE
- Str8Tween.pdf

Namespace

The different Str8Tween parts are wrapped in a single namespace which is Str8lines.Tweening.

It has to be added as follow at the top of your C# scripts in order to access Str8Tween engine classes :

```
use Str8lines.Tweening;
```

Functions

The interesting functions are located in the `Str8Tween` and `Tween` classes.

Others classes may have public functions but their use has no interest if you just want to use Str8Tween normally.

Str8Tween

There are two types of methods available.

The ones used to create tweens are named after the effect you want to apply to a target (e.g. : move, rotate, fade).

Any other method in `Str8Tween` is meant to get existing tweens and manage them.

You can checkout every function available in Str8Tween in the [Scripting Api](#).

Tween

The methods belonging to the `Tween` class allows to instantiate, control, and add callbacks on the events of this class instances.

Since `Str8Tween` class handles tween's instantiation and other operations at the same time, it's highly

recommended to use `Str8Tween` methods instead of tween's constructors.

Also, the `update()` function being automatically called by the `TweensHandler` class at each frame, there's is no need for a user to call it if he just uses `Str8Tween` as it is.

Additional resources

- [Str8Tween](#) *scripting api class reference*
- [Tween](#) *scripting api class reference*
- [TweensHandler](#) *scripting api class reference*
- [AssetPackagesImport](#) *external resource*

Create a Tween

To create a tween call the `Str8tween`'s functions named after the possible effects. The engine stores a reference to the instance until its destruction. This way you can manage your tweens later on.

Parameters

The instantiation methods takes four mandatory parameters and an optional one.

```
Str8Tween.move(target's component, end value, easing function, duration);
```

In order :

- The target's component is a Unity component (e.g. `RectTransform`, `Graphic`...).
- The end value is a `float` or a `Vector3` depending on the function called.
- The easing function specifies the rate of value change over time.
- The duration in seconds.
- An optional boolean to define if the tween is automatically destroyed when it ends. If `false` you will be able to reuse it.

Delay and Loops

You can add a delay to a tween as follow :

```
someTween.delay(1f);
```

The `float` passed in parameters represents the time to wait in seconds before the tween starts.

To make a tween repeat itself, pass the number of loops to make and the loop style in the arguments of the loop function :

```
someTween.loop(3, LoopStyle.Oscillate);
```

Both arguments are optional. The default number of loops is -1 (infinite loops) and the default loop style is `LoopStyle.Restart`.

Additional resources

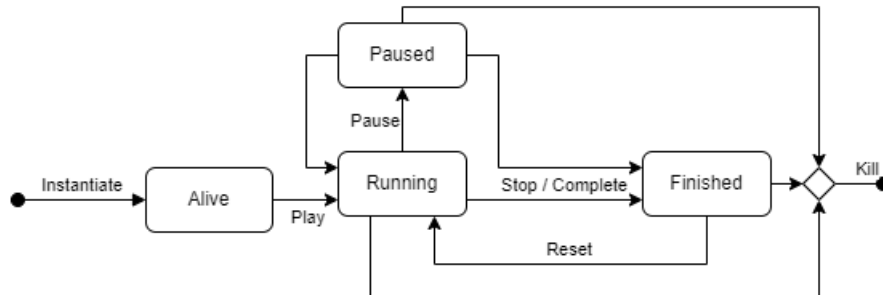
- [Effects](#) *get started guide reference*
- [Timing](#) *get started guide reference*
- [EasingFunction](#) *scripting api class reference*
- [LoopStyle](#) *scripting api class reference*
- [RectTransform](#) *external resource*
- [Graphic](#) *external resource*
- [Vector3](#) *external resource*

Tweens Management

Here we will see how a tween behaviors by default and how we can control it along its lifecycle.

Lifecycle

Before anything, here is the representation of a tween's lifecycle created within the engine :



Once a tween is instantiated it plays automatically and while it's alive any control function can be called. Killing a tween sets its `isAlive` property to `false`, making `Str8Tween` loose track of it leading to its destruction. Until it dies, you can reset a tween at any moment.

Controls

The following functions can be accessed from the tween instances :

- `play()`
- `pause()`
- `stop()`
- `complete()`
- `reset()`
- `kill()`

The name of the functions being self explanatory we will look at some important details.

The `stop()` and `complete()` methods both have an optional parameter called `triggerOnEnd` (a boolean) which allows you to raise or not the end event of a tween.

The `complete()` method also have a second optional parameter which is a `CompletionMode` allowing you to define the way of handling the final value of looping tweens.

Using the `Str8Tween` class you can control one or more tweens with a single method call.

Indeed, the previously listed functions are also available in this class.

They can take in parameter a tween's `id` to impact the corresponding one only or a `GameObject` to affect every tweens having the same target.

If you don't set any parameter all tweens will be impacted.

Finally you can fetch one or more tweens using the `get()` method which can also have an `id` or a `GameObject` as parameter or no parameter at all.

interactions

You have the possibility to interact with the tweens you make through their start, loop and end events.

Use the `onStart()`, `onLoop()` and `onEnd()` functions to define the callbacks you want to be fired at these events.

The example below shows that you can use anonymous functions or classic methods.

```
Tween t = Str8Tween.move(target, end value, easing function, duration);
t.onStart(()=>{ Debug.Log("Hi mom !"); })
.onLoop(loopCallback);

private void loopCallback(int loopsCount)
{
    string message = "Loops accomplished : " + loopsCount;
    Debug.Log(message);
}
```

Additional resources

- [Tween](#) *scripting api class reference*
- [CompletionMode](#) *scripting api class reference*
- [Str8Tween](#) *scripting api class reference*
- [Controls](#) *get started guide reference*
- [Events](#) *get started guide reference*
- [GameObject](#) *external resource*

Examples

Simple tween

```
using UnityEngine;
using Str8lines.Tweening;

public class Example : MonoBehaviour
{
    public GameObject target;

    private void Start()
    {
        RectTransform rect = target.GetComponent<RectTransform>();
        Vector3 endValue = new Vector3(rect.anchoredPosition3D.x + 300f, rect.anchoredPosition3D.y,
rect.anchoredPosition3D.z);

        Tween t = Str8Tween.move(rect, endValue, EasingFunction.Linear, 3f);
        t.onStart(()=>{ _onStart(t); });
    }

    private void _onStart(Tween t)
    {
        string message = "Tween (" + t.id + ") is started";
        Debug.Log(message);
    }
}
```

Tweens chaining


```

using UnityEngine;
using UnityEngine.UI;
using Str8lines.Tweening;

public class Example : MonoBehaviour
{
    public GameObject target;
    private string _tweenID;

    private void Start()
    {
        RectTransform rect = target.GetComponent<RectTransform>();
        Image image = target.GetComponent<Image>();
        Vector3 endValue = new Vector3(2, 2, 2);

        Tween scaleTween = Str8Tween.scale(rect, endValue, EasingFunction.OutBack, 1.75f);
        scaleTween.onEnd(()=>{
            Tween fadeTween = Str8Tween.fade(image, 0f, EasingFunction.OutQuint, 2f)
                .loop(-1, LoopStyle.Oscillate)
                .onLoop((loopsCount)=>{
                    Debug.Log("Loops accomplished => " + loopsCount);
                });
            fadeTween.onStart(()=>{ _onStart(fadeTween); });
        });

    private void _onStart(Tween t)
    {
        _tweenID = t.id;
        string message = "Tween (" + _tweenID + ") is started";
        Debug.Log(message);
    }

    private void Update()
    {
        //Press spacebar to end le looping tween
        if(Input.GetKeyDown(KeyCode.Space))
        {
            Tween fetched = Str8Tween.get(_tweenID);
            fetched.stop();
        }
    }
}

```

Multiple tweens

```
using UnityEngine;
using Str8lines.Tweening;

public class Example : MonoBehaviour
{
    public GameObject target;

    private void Start()
    {
        RectTransform rect = target.GetComponent<RectTransform>();
        Vector3 endValue = new Vector3(rect.anchoredPosition3D.x + 300f, rect.anchoredPosition3D.y,
rect.anchoredPosition3D.z);
        Tween moveTween = Str8Tween.move(rect, endValue, EasingFunction.Linear, 1.5f);

        endValue = new Vector3(rect.localEulerAngles.x, rect.localEulerAngles.y, rect.localEulerAngles.z + 360f);
        Tween rotateTween = Str8Tween.rotate(rect, endValue, EasingFunction.OutBack, 1.5f).delay(0.2f);
    }
}
```

Effects

Four target related effects are available to be applied on 2D elements.

Here is a description of these effects and the possibilities they offer with some illustrations.

Target related effects

In this section the effects covered shows the different properties that a tween can update on a target.

move

Does a translations of the target.

It updates the `anchoredPosition3D` property of the `RectTransform`.

As the end value to give is a `Vector3`, you can play with the x and y values to change translation's direction. The z value may not be as interesting for this operation.



scale

This effect is self-explanatory. It updates the `localScale` property of the `RectTransform`.

Just like the move effect the interesting values to play with are the x and y ones, not the z value.



rotate

Also self-explanatory.

It updates the `localEulerAngles` property of the `RectTransform`.

Contrary to the previous effects, the x, y and z values are interesting to work with.



fade

Changes the transparency of the targeted `CanvasRenderer`, `SpriteRenderer` or `Graphic` component.

The property updated is always the alpha of the component.

Since `Image`, `RawImage` and `Text` components inherit from `Graphic` you can pass these components in the function's parameters.



Loops

Make a tween loop is a tween related effect you can apply.

To use it, call the `loop()` method.

There are three styles of loop that can be used having a different visual result :

- `LoopStyle.Restart` : The tween will always go from the starting value to the defined end value.

A

B



- `LoopStyle.Oscillate` : The tween is played alternatively forward and backward.

A

B



- `LoopStyle.WithOffset` : At the end of every loop, the end value becomes the starting value of the next loop. The end value is recalculated to apply the same target related effect from the new starting value.

A

B

C

D



Additional resources

- [LoopStyle](#) *scripting api class reference*
- [RectTransform](#) *external resource*
- [Graphic](#) *external resource*
- [CanvasRenderer](#) *external resource*
- [SpriteRenderer](#) *external resource*
- [Image](#) *external resource*
- [RawImage](#) *external resource*
- [Text](#) *external resource*
- [Vector3](#) *external resource*

Timing

The `duration` of a tween is not the only way of handling your animations timing. Moreover it's a notion that can be defined in two ways depending on the context.

Loops duration

Using the `loop()` method, the `duration` settled for the tween becomes the duration of each loop. You can define an infinite number of loops by setting the loop count to -1 or define a finite number of loops. The total duration of the tween then becomes the number of loops times the duration of one single loop. If you need to know the time elapsed you can consult the following two properties of a tween :

- `elapsedTotal`
- `elapsedSinceDelay`

Delay

A delay can be added by calling the `delay()` function after tween creation. This function takes an argument which represents the time to pass in seconds before the tween starts. This parameter is a `float` that can not be negative or equal to zero. In case you make a tween loop, the delay you set is applied only at the first loop.

Additional resources

- [Tween](#) *scripting api class reference*

Events

Three types of event with their specificities are available.
Let's have a look to them.

Start

The start event is raised on the very beginning of the tween.
By that I mean that it's raised after any delay you would have set.
To add a callback on this event you need to use the `onStart()` function of a tween.

Loop

The loop event is raised at every loop end, including the last one.
So when a tween ends naturally, both loop and end event are raised.
The `completedLoopsCount` is returned by the event so you can easily use it in your callbacks.
To add a callback use the `onLoop()` function of a Tween.

End

The end event may be the most usefull one. It's raised when the tween finishes.
When the tween is ended by a call of the `stop()` or the `complete()` method you can choose to avoid the raise of this event and thus the callbacks trigger.
To add a callback to the end event use the `onEnd()` function of a Tween.

Additional resources

- [Tween](#) *scripting api class reference*

Controls

In this part we won't talk about play, pause and kill functions as their names are self explanatory. But we will talk about the different ways of ending a tween and how resetting them can behave differently depending on the context.

Completion Mode

Str8Tween introduces what is called a completion mode.

It determines the different ways of handling the end value of a tween when it finishes.

Off course when a tween ends naturally or is not looping nothing happens in regard to the end value, in this case the completion mode have no impact.

In fact it becomes interesting only when you look at looping tweens.

Let's illustrate with the case of a tween looping using `LoopStyle.WithOffset` as an example.

We can imagine three different end values :

- The value to reach given by the user.
- The value to reach plus the offset added at every loop.
- The value you would have reached if the tween ended naturally. The completion mode was implemented to solve this issue of multiple expectable possible end values.

There are three completion modes :

- `CompletionMode.Static` : locks the given value as the initial end value.

A

B

C

D



- `CompletionMode.Dynamic` : updates the end value after each loop in order to take in count the offset in the end value.

A

B

C

D



- `CompletionMode.Projectend` : makes the tween end on the value it would have reached if it ended naturally after its finite number of loops.

A

B

C

D



How does these modes behave when using other loop styles ?

With `LoopStyle.Oscillate` :

- `CompletionMode.Static` : does exactly the same thing.
- `CompletionMode.Dynamic` : alternatly finish on the end or the start value, depending on the tween playing forward or backward.
- `CompletionMode.Projecte` : finish on the end or the start value, depending on the number of loops being odd or even.

With `LoopStyle.Restart` the final value will always be the end value defined by the user.

Stop VS Complete

The differences between these two methods are a bit deeper than having different end values for the tween. The following paragraphs details the methods role and may allow you to wisely choose which one to use when.

Stop method

It makes the tween finish but does not kill it. When calling the function you can choose to trigger the end event or not.

The final value of the tween is the value it had at the moment of calling the method.

To simplify it is visually the same as pausing the tween but ends the tween.

Complete method

Technically it calls the stop method and handles the end value changes so you also have the possibility to trigger the end event or not.

The end value is managed to visually reproduce the natural finish of the tween, but a "natural end" can be perceived differently from one user to another.

Moreover the "natural" end of the tween may not fit with what you want to achieve in your project in practice.

This is why we introduced a completion mode.

Reset

You are able to reset a tween at any time so there are multiple situations where a tween can be when you do it.

Notice that the delay is always replayed when you reset the tween. If you want to avoid this behaviour you have to manually set the delay to 0 seconds before you reset the tween.

Also, resetting a tween does not change the fact that it was running or not.

Thus, if the tween was paused it is still paused after reset and you will have to use the `play()` method.

In contrary, if the tween was playing it keeps playing after the reset.

If the tween is finished, the default behaviour is that the tween won't be running on reset.

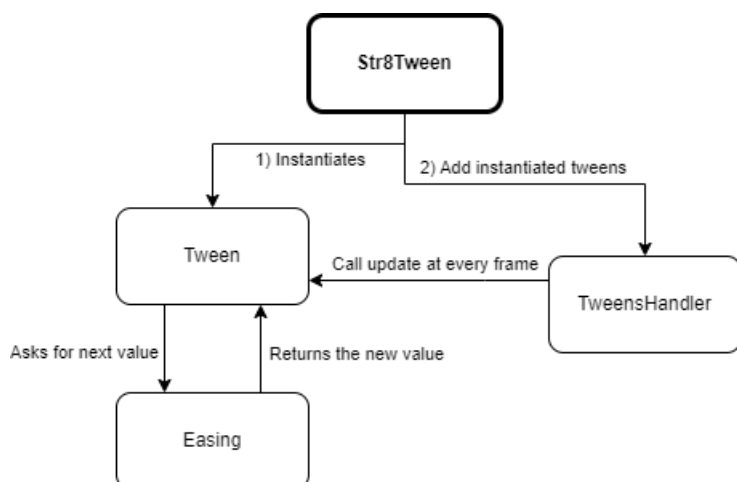
Despite this, you can directly call the `play()` function, as the tween is finished, it will automatically reset the tween then play it.

Additional resources

- [Tween scripting api class reference](#)
- [CompletionMode scripting api class reference](#)

Global Functionning

Here you will find the different classes contained in Str8Tween.dll, a description of their role and what they are designed to do.



Str8tween

This is the main class of the tween engine. It is responsible for tween instantiation, fetch and control. The fetch and control methods are designed to aim one or more tween at the time. Once a tween is created it sends the tween to the **TweensHandler**.

Tween

The tween class holds the data used for the calculations and the rules conditioning the lifecycle. It performs the values update on the defined target. The update function is public so it can be called by the **TweensHandler** class at every frame. This class also have the methods to control them individually and to define their events callbacks. It encloses properties representing an id, the parameters used by the constructors and other data in regards to its lifecycle.

TweensHandler

It's a singleton used to maintain the collection of alive tweens and update them at each frame if they are alive, if not it removes them from the collection. The only method available is meant to be used by the **Str8Tween** class only so it can add the tweens it creates to the collection.

Easing

The easing class contains the implementations of Robert Penner's equations. The ease methods returns the result of the equations. They are used to calculate the values to set based on the time elapsed since the previous frame.

Additional resources

- [Str8Tween scripting api class reference](#)
- [Tween scripting api class reference](#)
- [TweensHandler scripting api class reference](#)
- [Easing scripting api class reference](#)

- [Robert Penner's easing equations](#) *external resource*

Properties

Every tween have a set of readonly properties. We could separate those in three categories.

Parameters

They are what defines the identity of the tween and how it should work.

Here we will discuss of just a few.

- `id` : It is unique and randomly generated on instantiation. `Str8Tween` uses it to fetch a specific tween.
- `target` : It can not be null. If it ever happens the `TweensHandler` class kills the tween.
- `duration` : It's the value settled by the user.
- `loopsCount` : There is no dedicated property to know if a tween is looping or not. But the default value on tween instantiation is 0. Any tween looping will have this property equal to another value.

Status

We can count three of them :

- `isAlive`
- `isFinished`
- `isRunning`

They define where the tween is in its lifecycle.

Note that you may not see the `isAlive` property of a tween set to `false` as it should be automatically destroyed by the engine in this case.

To deepen the subject about tweens lifecycle please refer to lifecycle paragraph in the Get Started guide.

Data

There are also three properties of this kind.

- `elapsedTotal`
- `elapsedSinceDelay`
- `completedLoopsCount`

They are all related to time, even the `completedLoopsCount` which helps you locate in time the tween concerned.

Each frame, Unity returns the time elapsed since the previous one.

Every tween records this data only when it is running.

Additional resources

- [Tween scripting api class reference](#)
- [Tween's lifecycle get started guide reference](#)
- [Time.deltaTime external resource](#)

Chaining Tweens

Str8Tween does not provide a simplified way of creating a sequence of tweens.

You will have to use the `onEnd()` method to chain tweens.

This way of chaining your tweens may lead to a callback hell like shown below.

```
public GameObject target;

private void Start()
{
    RectTransform rect = target.GetComponent<RectTransform>();
    Image image = target.GetComponent<Image>();
    Vector3 initialAngles = target.transform.localEulerAngles;
    Vector3 endAngles = new Vector3(initialAngles.x, initialAngles.y, initialAngles.z + 360f);

    Str8Tween.scale(rect, 2 * Vector3.one, EasingFunction.OutBack, 1.75f)
        .onEnd(()=>{
            Str8Tween.fade(image, 0.5f, EasingFunction.OutQuint, 2f)
                .onEnd(()=>{
                    Str8Tween.rotate(rect, endAngles, EasingFunction.OutBack, 1.5f)
                        .onEnd(()=>{
                            Str8Tween.fade(image, 1f, EasingFunction.OutQuint, 2f)
                                .onEnd(()=>{
                                    Str8Tween.scale(rect, Vector3.one, EasingFunction.OutBack, 1.75f)
                                        .onEnd(()=>{
                                            Debug.Log("Chain is over !");
                                        });
                                    });
                                });
                            });
                        });
                    });
                });
            });
        });
    });
}
```

To bring a simple way to safely chain tweens is something delicate inside Str8Tween that requires a bit of work.

If users are interested into a feature dedicated to tweens chaining in an easier way, I could eventually implement that in futur version.

Namespace Str8lines.Tweening

Classes

Easing

Defines easing functions used internally and provides methods to calculate [Vector3](#) and `float` values.

Str8Tween

Str8Tween is a tween engine. This class is meant to instantiate [tweens](#) and manage their lifecycle. It provides additional methods to manipulate every [tweens](#) created.

Tween

Representation of a tween. A uuid is attributed to the tween on creation. Each instances can be manipulated with a variety of methods.

TweensHandler

A singleton used to keep track of every [Tween](#) created with [Str8Tween](#). Updates alive ones and removes the dead ones on every frame.

Enums

CompletionMode

Defines end values to use after the tween completion.

EasingFunction

Used to specify the rate of change over time. See easings.net to visualize each easing function.

LoopStyle

Defines if values are reset on loop (Restart), if the tween is played forward then backward (Oscillate) or if tweening restarts from end values (WithOffset).

Delegates

Tween.TweenDelegate

Delegate for start and complete events.

Tween.TweenLoopDelegate

Delegate for loop event.

Class Easing

Defines easing functions used internally and provides methods to calculate [Vector3](#) and `float` values.

Methods

ease(EasingFunction, Single, Single, Single, Single)

Calculates new `float` value according to elapsed time.

Parameters

TYPE	NAME	DESCRIPTION
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	t	Time elapsed (in seconds).
Float	f	Initial <code>float</code> value.
Float	c	<code>float</code> value change.
Float	d	Total tween duration (in seconds).

Returns

TYPE	DESCRIPTION
Float	Resulting <code>float</code> value.

Examples

Calculates new float value.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private float val = 0f;

    private void Start()
    {
        val = Easing.ease(EasingFunction.Linear, 2f, val, 10f, 3f);
    }
}
```

ease(EasingFunction, Single, Vector3, Vector3, Single)

Calculates new [Vector3](#) according to elapsed time.

Parameters

TYPE	NAME	DESCRIPTION
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	t	Time elapsed (in seconds).
Vector3	v	Initial Vector3 value.
Vector3	c	Vector3 value change.
Float	d	Total tween duration (in seconds).

Returns

TYPE	DESCRIPTION
Vector3	Resulting Vector3 .

Examples

Calculates new Vector3 value.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public Vector3 vector;

    private void Start()
    {
        Vector3 vectorChange = new Vector3(0, 500, 0);
        vector = Easing.ease(EasingFunction.Linear, 2f, vector, vectorChange, 3f);
    }
}
```

Class Str8Tween

Str8Tween is a tween engine. This class is meant to instantiate [tweens](#) and manage their lifecycle. It provides additional methods to manipulate every [tweens](#) created.

Methods

complete(GameObject, Boolean, CompletionMode)

Completes every [tween](#) associated to the given [GameObject](#).

Parameters

TYPE	NAME	DESCRIPTION
GameObject	target	The target of the tweens to complete .
Boolean	triggerOnEnd	(Optional) If <code>true</code> , triggers tween 's end event. Default value is <code>true</code>
CompletionMode	mode	(Optional) The completion mode defines the end values to apply. Default value is <code>STATIC</code>

Examples

Press space to complete target's tweens :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public GameObject target;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.complete(target);
    }
}
```

complete(Boolean, CompletionMode)

Completes every [tween](#) alive. See also [complete\(Boolean, CompletionMode\)](#).

Parameters

TYPE	NAME	DESCRIPTION
Boolean	triggerOnEnd	(Optional) If <code>true</code> , triggers tween 's end event. Default value is <code>true</code>
CompletionMode	mode	(Optional) The completion mode defines the end values to apply. Default value is <code>STATIC</code>

Examples

Press space to complete every tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.complete();
    }
}
```

complete(String, Boolean, CompletionMode)

Completes the [tween](#) associated to the given uuid. See also [complete\(Boolean, CompletionMode\)](#).

Parameters

TYPE	NAME	DESCRIPTION
String	id	The id of the tween to complete.
Boolean	triggerOnEnd	(Optional) If <code>true</code> , triggers tween 's end event. Default value is <code>true</code>
CompletionMode	mode	(Optional) The completion mode defines the end values to apply. Default value is <code>STATIC</code>

Examples

Press space to complete a tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.complete("7aac6207-107f-4ddc-8761-122f669d3e3b");
    }
}
```

fade(CanvasRenderer, Single, EasingFunction, Single, Boolean)

Instantiates a new [tween](#) which changes `canvasRenderer`'s alpha to a given value.

Parameters

TYPE	NAME	DESCRIPTION
CanvasRenderer	canvasRenderer	The CanvasRenderer which will have its alpha changed.
Float	toValue	A <code>float</code> that represents <code>canvasRenderer</code> 's final alpha value.

TYPE	NAME	DESCRIPTION
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed on end.

Returns

TYPE	DESCRIPTION
Tween	The tween created.

Examples

Changes the CanvasRenderer's alpha during 3 seconds with a linear easing.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public CanvasRenderer canvasRenderer;

    private void Start()
    {
        Str8Tween.fade(canvasRenderer, 0f, EasingFunction.Linear, 3f);
    }
}
```

fade(Graphic, Single, EasingFunction, Single, Boolean)

Instantiates a new [tween](#) which changes `graphic`'s alpha to a given value.

Parameters

TYPE	NAME	DESCRIPTION
Graphic	graphic	The Graphic which will have its alpha changed.
Float	toValue	A <code>float</code> that represents <code>graphic</code> 's final alpha value.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).

--	--	--

TYPE	NAME	DESCRIPTION
Boolean	killOnEnd	(Optional) If <code>true</code> , the <code>tween</code> will be destroyed on end.

Returns

TYPE	DESCRIPTION
<code>Tween</code>	The <code>tween</code> created.

Examples

Changes the Graphic's alpha during 3 seconds with a linear easing.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public Graphic graphic;

    private void Start()
    {
        Str8Tween.fade(graphic, 0f, EasingFunction.Linear, 3f);
    }
}
```

fade(Image, Single, EasingFunction, Single, Boolean)

Instantiates a new `tween` which changes `image`'s alpha to a given value.

Parameters

TYPE	NAME	DESCRIPTION
Image	image	The <code>Image</code> which will have its alpha changed.
Float	toValue	A <code>float</code> that represents <code>image</code> 's final alpha value.
<code>EasingFunction</code>	easingFunction	The <code>easing function</code> represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the <code>tween</code> will be destroyed on end.

Returns

TYPE	DESCRIPTION
Tween	The tween created.

Examples

Changes the Image's alpha during 3 seconds with a linear easing.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public Image image;

    private void Start()
    {
        Str8Tween.fade(image, 0f, EasingFunction.Linear, 3f);
    }
}
```

fade(RawImage, Single, EasingFunction, Single, Boolean)

Instantiates a new [tween](#) which changes `rawImage`'s alpha to a given value.

Parameters

TYPE	NAME	DESCRIPTION
RawImage	rawImage	The RawImage which will have its alpha changed.
Float	toValue	A <code>float</code> that represents <code>rawImage</code> 's final alpha value.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed on end.

Returns

TYPE	DESCRIPTION
Tween	The tween created.

Examples

Changes the RawImage's alpha during 3 seconds with a linear easing.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RawImage rawImage;

    private void Start()
    {
        Str8Tween.fade(rawImage, 0f, EasingFunction.Linear, 3f);
    }
}
```

fade(SpriteRenderer, Single, EasingFunction, Single, Boolean)

Instantiates a new **tween** which changes `spriteRenderer`'s alpha to a given value.

Parameters

TYPE	NAME	DESCRIPTION
SpriteRenderer	spriteRenderer	The SpriteRenderer which will have its alpha changed.
Float	toValue	A <code>float</code> that represents <code>spriteRenderer</code> 's final alpha value.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed on end.

Returns

TYPE	DESCRIPTION
Tween	The tween created.

Examples

Changes the SpriteRenderer's alpha during 3 seconds with a linear easing.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public SpriteRenderer spriteRenderer;

    private void Start()
    {
        Str8Tween.fade(spriteRenderer, 0f, EasingFunction.Linear, 3f);
    }
}
```

fade(Text, Single, EasingFunction, Single, Boolean)

Instantiates a new **tween** which changes `text`'s alpha to a given value.

Parameters

TYPE	NAME	DESCRIPTION
Text	text	The Text which will have its alpha changed.
Float	toValue	A <code>float</code> that represents <code>text</code> 's final alpha value.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed on end.

Returns

TYPE	DESCRIPTION
Tween	The tween created.

Examples

Changes the Text's alpha during 3 seconds with a linear easing.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public Text text;

    private void Start()
    {
        Str8Tween.fade(text, 0f, EasingFunction.Linear, 3f);
    }
}
```

get()

Returns the **tweens** which are still alive.

Returns

TYPE	DESCRIPTION
Tween[]	An Array of tweens .

Examples

Displays the ids of every existing tween in console :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Start()
    {
        Tween[] tweens = Str8Tween.get();
        if(tweens.Length > 0){
            foreach(Tween t in tweens){
                UnityEngine.Debug.Log(t.id)
            }
        }
    }
}
```

get(GameObject)

Returns the **tweens** associated to the given **GameObject**.

Parameters

TYPE	NAME	DESCRIPTION
GameObject	target	The target of the tweens to retrieve.

Returns

TYPE	DESCRIPTION
Tween[]	An Array of tweens .

Examples

Displays target's tweens ids in console :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public GameObject target;

    private void Start()
    {
        Tween[] tweens = Str8Tween.get(target);
        if(tweens.Length > 0){
            foreach(Tween t in tweens){
                UnityEngine.Debug.Log(t.id)
            }
        }
    }
}
```

get(String)

Returns the [tween](#) associated to the given uuid.

Parameters

TYPE	NAME	DESCRIPTION
String	id	The id of the tween to retrieve.

Returns

TYPE	DESCRIPTION
Tween	The tween retrieved or <code>null</code> if not found.

Examples

Gets a tween from its id :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Start()
    {
        Tween t = Str8Tween.get("7aac6207-107f-4ddc-8761-122f669d3e3b");
    }
}
```

kill()

Kills every [tween](#) alive. See also [kill\(\)](#).

Examples

Press space to kill every tween :


```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.kill();
    }
}
```

kill(GameObject)

Kills every [tween](#) associated to the given [GameObject](#).

Parameters

TYPE	NAME	DESCRIPTION
GameObject	target	The target of the tweens to kill .

Examples

Press space to kill target's tweens :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public GameObject target;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.kill(target);
    }
}
```

kill(String)

Kills the [tween](#) associated to the given uuid. See also [kill\(\)](#).

Parameters

TYPE	NAME	DESCRIPTION
String	id	The id of the tween to kill.

Examples

Press space to kill a tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.kill("7aac6207-107f-4ddc-8761-122f669d3e3b");
    }
}
```

move(RectTransform, Vector3, EasingFunction, Single, Boolean)

Instantiates a new [tween](#) which changes the [anchoredPosition3D](#) of the `rectTransform`.

Parameters

TYPE	NAME	DESCRIPTION
RectTransform	rectTransform	The RectTransform to move.
Vector3	toValue	A Vector3 that represents <code>rectTransform</code> 's final position.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed on end.

Returns

TYPE	DESCRIPTION
Tween	The tween created.

Examples

Changes the `rectTransform`'s `anchoredPosition3D` during 3 seconds in a linear motion.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500);
        Str8Tween.move(rectTransform, destination, EasingFunction.Linear, 3f);
    }
}
```

pause()

Pauses every [tween](#) alive. See also [pause\(\)](#).

Examples

Press space to pause existing tweens :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.pause();
    }
}
```

pause(GameObject)

Pauses every [tween](#) associated to the given [GameObject](#).

Parameters

TYPE	NAME	DESCRIPTION
GameObject	target	The target of the tweens to pause .

Examples

Press space to pause target's tweens :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public GameObject target;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.pause(target);
    }
}
```

pause(String)

Pauses the [tween](#) associated to the given uuid. See also [pause\(\)](#).

Parameters

TYPE	NAME	DESCRIPTION
String	id	The id of the tween to pause.

Examples

Press space to pause a tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.pause("7aac6207-107f-4ddc-8761-122f669d3e3b");
    }
}
```

play()

Plays every [tween](#) alive. See also [play\(\)](#).

Examples

Press space to play existing tweens :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.play();
    }
}
```

play(GameObject)

Plays every [tween](#) associated to the given [GameObject](#).

Parameters

TYPE	NAME	DESCRIPTION
GameObject	target	The target of the tweens to play .

Examples

Press space to play target's tweens :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public GameObject target;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.play(target);
    }
}
```

play(String)

Plays the [tween](#) associated to the given uuid. See also [play\(\)](#).

Parameters

TYPE	NAME	DESCRIPTION
String	id	The id of the tween to play.

Examples

Press space to play a tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.play("7aac6207-107f-4ddc-8761-122f669d3e3b");
    }
}
```

reset()

Reset every [tween](#) alive. See also [reset\(\)](#).

Examples

Press space to reset every tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.reset();
    }
}
```

reset(GameObject)

Reset every [tween](#) associated to the given [GameObject](#).

Parameters

TYPE	NAME	DESCRIPTION
GameObject	target	The target of the tweens to reset .

Examples

Press space to reset target's tweens :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public GameObject target;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.reset(target);
    }
}
```

reset(String)

Resets the [tween](#) associated to the given uuid.

Parameters

TYPE	NAME	DESCRIPTION
String	id	The id of the tween to reset. See also reset()

Examples

Press space to reset a tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.reset("7aac6207-107f-4ddc-8761-122f669d3e3b");
    }
}
```

rotate(RectTransform, Vector3, EasingFunction, Single, Boolean)

Instantiates a new [tween](#) which changes the [localEulerAngles](#) of the `rectTransform`.

Parameters

TYPE	NAME	DESCRIPTION
RectTransform	rectTransform	The RectTransform that will rotate.
Vector3	toValue	A Vector3 that represents <code>rectTransform</code> 's final rotation.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).

TYPE	NAME	DESCRIPTION
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed on end.

Returns

TYPE	DESCRIPTION
Tween	The tween created.

Remarks

Due to the way Unity handles rotations, `toValue` may differ from `rectTransform`'s rotation in the inspector. To learn more about rotations in Unity, please check : [Rotation and Orientation in Unity](#).

Examples

Changes the `rectTransform`'s `localEulerAngles` during 3 seconds in a linear motion.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 newAngles = new Vector3(90, 180, 0);
        Str8Tween.rotate(rectTransform, newAngles, EasingFunction.Linear, 3f);
    }
}
```

scale(RectTransform, Vector3, EasingFunction, Single, Boolean)

Instantiates a new [tween](#) which changes the [localScale](#) of the `rectTransform`.

Parameters

TYPE	NAME	DESCRIPTION
RectTransform	rectTransform	The RectTransform to scale.
Vector3	toValue	A Vector3 that represents <code>rectTransform</code> 's final scale.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed on end.

Returns

TYPE	DESCRIPTION
Tween	The tween created.

Examples

Changes rectTransform's localScale during 3 seconds with a linear easing.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 newScale = new Vector3(2, 2, 2);
        Str8Tween.scale(rectTransform, newScale, EasingFunction.Linear, 3f);
    }
}
```

stop(GameObject, Boolean)

Stops every [tween](#) associated to the given [GameObject](#).

Parameters

TYPE	NAME	DESCRIPTION
GameObject	target	The target of the tweens to stop .
Boolean	triggerOnEnd	

Examples

Press space to stop target's tweens :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public GameObject target;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.stop(target);
    }
}
```

stop(Boolean)

Stops every [tween](#) alive. See also [stop\(Boolean\)](#).

Parameters

TYPE	NAME	DESCRIPTION
Boolean	triggerOnEnd	

Examples

Press space to stop every tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.stop();
    }
}
```

stop(String, Boolean)

Stops the [tween](#) associated to the given uuid. See also [stop\(Boolean\)](#).

Parameters

TYPE	NAME	DESCRIPTION
String	id	The id of the tween to stop.
Boolean	triggerOnEnd	

Examples

Press space to stop a tween :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) Str8Tween.stop("7aac6207-107f-4ddc-8761-122f669d3e3b");
    }
}
```

Class Tween

Representation of a tween. A uuid is attributed to the tween on creation. Each instances can be manipulated with a variety of methods.

Constructors

Tween(CanvasRenderer, Single, EasingFunction, Single, Boolean, String)

Instantiate a new tween, initialize it and give it a UUID.

Parameters

TYPE	NAME	DESCRIPTION
CanvasRenderer	canvasRenderer	The CanvasRenderer on which changes will be applied.
Float	toValue	A <code>float</code> that represents <code>canvasRenderer</code> 's final alpha.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed once completed. Default value is <code>true</code>
String	methodName	Name of the method which called this constructor.

Examples

Create new tween that changes CanvasRenderer's alpha :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public CanvasRenderer canvasRenderer;

    private void Start()
    {
        Tween t = new Tween("fade", canvasRenderer, 0f, EasingFunction.Linear, 3f);
    }
}
```

Tween(Graphic, Single, EasingFunction, Single, Boolean, String)

Instantiate a new tween, initialize it and give it a UUID.

Parameters

TYPE	NAME	DESCRIPTION
Graphic	graphic	The Graphic on which changes will be applied.
Float	toValue	A <code>float</code> that represents <code>graphic</code> 's final alpha.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed once completed. Default value is <code>true</code>
String	methodName	Name of the method which called this constructor.

Examples

Create new tween that changes graphic's alpha :

```

using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public Graphic graphic;

    private void Start()
    {
        Tween t = new Tween("fade", graphic, 0f, EasingFunction.Linear, 3f);
    }
}

```

Tween(RectTransform, Vector3, EasingFunction, Single, Boolean, String)

Instantiate a new tween, initialize it and give it a UUID.

Parameters

TYPE	NAME	DESCRIPTION
RectTransform	rectTransform	The RectTransform on which changes will be applied.
Vector3	toVector	A Vector3 that represents <code>rectTransform</code> 's final position, scale or rotation.
EasingFunction	easingFunction	The easing function represents the type of easing.

TYPE	NAME	DESCRIPTION
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed once completed. Default value is <code>true</code>
String	methodName	Name of the method which called this constructor.

Examples

Create new tween that changes target's position :

```

using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        Tween t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
    }
}

```

Tween(SpriteRenderer, Single, EasingFunction, Single, Boolean, String)

Instantiate a new tween, initialize it and give it a UUID.

Parameters

TYPE	NAME	DESCRIPTION
SpriteRenderer	spriteRenderer	The SpriteRenderer on which changes will be applied.
Float	toValue	A <code>float</code> that represents <code>spriteRenderer</code> 's final alpha.
EasingFunction	easingFunction	The easing function represents the type of easing.
Float	duration	Total tween duration (in seconds).
Boolean	killOnEnd	(Optional) If <code>true</code> , the tween will be destroyed once completed. Default value is <code>true</code>

--	--	--

TYPE	NAME	DESCRIPTION
String	methodName	Name of the method which called this constructor.

Examples

Create new tween that changes SpriteRenderer's alpha :

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public SpriteRenderer spriteRenderer;

    private void Start()
    {
        Tween t = new Tween("fade", spriteRenderer, 0f, EasingFunction.Linear, 3f);
    }
}
```

Properties

completedLoopsCount

Property Value

TYPE	DESCRIPTION
Int	The number of loops completed since the tween started.

duration

Property Value

TYPE	DESCRIPTION
Float	Duration of a tween's loop (in seconds). If the tween is not a loop then it's the duration of the tween itself.

easingFunction

Property Value

TYPE	DESCRIPTION
EasingFunction	Defines easing functions used internally and provides methods to calculate new values.

elapsedSinceDelay

Property Value

TYPE	DESCRIPTION
Float	Time elapsed in seconds (delay excluded).

elapsedTotal

Property Value

TYPE	DESCRIPTION
Float	Time elapsed in seconds (delay included).

id

Property Value

TYPE	DESCRIPTION
String	UUID given on creation.

isAlive

Property Value

TYPE	DESCRIPTION
Boolean	While <input type="checkbox"/> the tween is referenced in Str8Tween and can be controlled.

isFinished

Property Value

TYPE	DESCRIPTION
Boolean	If <input type="checkbox"/> the tween finished playing.

isRunning

Property Value

TYPE	DESCRIPTION
Boolean	If <input type="checkbox"/> the tween is currently playing.

killOnEnd

Property Value

TYPE	DESCRIPTION
Boolean	If <input type="checkbox"/> the tween will be killed after playing.

loopsCount

Property Value

TYPE	DESCRIPTION
Int	The number of loops to do.

loopStyle

Property Value

TYPE	DESCRIPTION
LoopStyle	The type of loop to use.

target

Property Value

TYPE	DESCRIPTION
GameObject	GameObject on which the tween will be applied.

Methods

complete(Boolean, CompletionMode)

Completes the tween.

Parameters

TYPE	NAME	DESCRIPTION
Boolean	triggerOnEnd	(Optional) If <code>true</code> , triggers the tween's end event. Default value is <code>true</code>
CompletionMode	mode	(Optional) The completion mode defines the end values to apply. Default value is <code>PROJECTED</code>

Remarks

Completing a tween sends the target to its final values.

Examples

Press space to complete.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;
    private Tween t;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) t.complete();
    }
}
```

delay()

Returns

TYPE	DESCRIPTION
Float	The delay before the tween starts (in seconds).

Examples

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        Tween t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
        Debug.Log(t.delay());
    }
}
```

delay(Single)

Add delay to the tween before it starts playing.

Parameters

TYPE	NAME	DESCRIPTION
Float	delay	Time before the tween start (in seconds).

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
Tween	The tween delayed.

Examples

Add 2 seconds of delay before the tween starts.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        Tween t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
        t.delay(2f);
    }
}
```

kill()

Kills the tween. This removes the tween's callbacks and sets [isAlive](#) to `false`. [Str8Tween](#) class will remove every reference to the tween.

Examples

Press space to kill.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;
    private Tween t;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) t.kill();
    }
}
```

loop(Int32, LoopStyle)

Makes the tween loop.

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
Int	loopsCount	(Optional) Number of loops to do. Default value is -1.
LoopStyle	loopStyle	(Optional) Type of loop. Default value is LoopStyle.Restart.

Returns

TYPE	DESCRIPTION
Tween	The tween which loops.

Remarks

Default loopsCount's value is -1 which is equivalent to infinite loops.

Examples

Realizes five oscillating loops of a move tween.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        Tween t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
        t.loop(5, LoopStyle.Oscillate);
    }
}
```

onEnd(Tween.TweenDelegate)

Adds a callback to the tween's end event.

Parameters

TYPE	NAME	DESCRIPTION
Tween.TweenDelegate	callback	Callback function to trigger.

Returns

TYPE	DESCRIPTION
Tween	The tween which will rise the end event.

Examples

Displays a message in console when the move tween ends.

```

using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        Tween t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
        t.onEnd(_onEnd);
    }

    private void _onEnd()
    {
        UnityEngine.Debug.Log("End");
    }
}

```

onLoop(Tween.TweenLoopDelegate)

Adds a callback to the tween's loop event.

Parameters

TYPE	NAME	DESCRIPTION
Tween.TweenLoopDelegate	callback	Callback function to trigger.

Returns

TYPE	DESCRIPTION
Tween	The tween which will rise the loop event.

Examples

Displays the number of loops accomplished in console when the move tween loops.

```

using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        Tween t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
        t.loop(5, LoopStyle.Oscillate).onLoop(_onLoop);
    }

    private void _onLoop(int loopsCount)
    {
        UnityEngine.Debug.Log("Loop count : " + loopsCount);
    }
}

```

onStart(Tween.TweenDelegate)

Adds a callback to the tween's start event.

Parameters

TYPE	NAME	DESCRIPTION
Tween.TweenDelegate	callback	Callback function to trigger.

Returns

TYPE	DESCRIPTION
Tween	The tween which will rise the start event.

Examples

Displays a message in console when the move tween starts.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        Tween t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
        t.onStart(_onStart);
    }

    private void _onStart()
    {
        UnityEngine.Debug.Log("Start");
    }
}
```

pause()

Pauses the tween.

Examples

Press space to pause.

```

using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;
    private Tween t;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) t.pause();
    }
}

```

play()

Plays the tween.

Examples

Press space to play.

```

using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;
    private Tween t;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) t.play();
    }
}

```

reset()

Resets the tween values.

Examples

Press space to reset.

```

using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;
    private Tween t;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) t.reset();
    }
}

```

stop(Boolean)

Stops the tween.

Parameters

TYPE	NAME	DESCRIPTION
Boolean	triggerOnEnd	(Optional) If <code>true</code> , triggers the tween's end event. Default value is <code>true</code>

Remarks

Stopping a tween does not change the target's current values.

Examples

Press space to stop.

```

using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;
    private Tween t;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) t.stop();
    }
}

```

update(Single)

Determines new values from the time `t` elapsed and applies it to the tween's `target`. Also triggers callbacks when required.

Parameters

TYPE	NAME	DESCRIPTION
Float	t	The time elapsed (in seconds).

Examples

Updates a tween at new frame.

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;
    private Tween t;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500, 0);
        t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
    }

    private void Update()
    {
        t.update(Time.deltaTime)
    }
}
```

Class TweenHandler

A singleton used to keep track of every Tween created with Str8Tween. Updates alive ones and removes the dead ones on every frame.

Properties

Instance

Property Value

TYPE	DESCRIPTION
TweenHandler	Returns the instance of the TweenHandler singleton.

tweens

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<String, Tween>	Returns the tweens stored in the class. The key is the tweens's UUID and the value is the Tween itself.

Methods

Add(Tween)

Adds the Tween passed in parameters to the tweens dictionary

Parameters

TYPE	NAME	DESCRIPTION
Tween	t	The Tween to add.

Examples

```
using UnityEngine;
using Str8lines.Tweening;

public class MyClass : MonoBehaviour
{
    public RectTransform rectTransform;

    private void Start()
    {
        Vector3 destination = new Vector3(0, 500);
        Tween t = new Tween("move", rectTransform, destination, EasingFunction.Linear, 3f);
        TweenHandler.Instance.Add(t);
    }
}
```


Enum CompletionMode

Defines end values to use after the tween completion.

Fields

NAME	DESCRIPTION
DYNAMIC	
PROJECTED	
STATIC	

Enum EasingFunction

Used to specify the rate of change over time. See easings.net to visualize each easing function.

Fields

NAME	DESCRIPTION
InBack	
InBounce	
InCirc	
InCubic	
InElastic	
InExpo	
InOutBack	
InOutBounce	
InOutCirc	
InOutCubic	
InOutElastic	
InOutExpo	
InOutQuad	
InOutQuart	
InOutQuint	
InOutSine	
InQuad	
InQuart	
InQuint	
InSine	
Linear	
OutBack	
OutBounce	

NAME	DESCRIPTION
OutCirc	
OutCubic	
OutElastic	
OutExpo	
OutQuad	
OutQuart	
OutQuint	
OutSine	

Enum LoopStyle

Defines if values are reset on loop (Restart), if the tween is played forward then backward (Oscillate) or if tweening restarts from end values (WithOffset).

Fields

NAME	DESCRIPTION
Oscillate	
Restart	
WithOffset	

Delegate Tween.TweenDelegate

Delegate for start and complete events.

Delegate Tween.TweenLoopDelegate

Delegate for loop event.

Parameters

TYPE	NAME	DESCRIPTION
Int	loopsCount	The number of loops completed.