Table of Contents

Overview

Getting started with JTween is a fairly straightforward process. It involves importing the plugin, adding any AssemblyDefnition references as needed, and importing its namespace into any script that you want to use the JTween API.

Step 1 - Importing JTween

Using this library in your project can be done in two ways:

- **Releases:** The latest release can be found here as a UnityPackage file that can be downloaded and imported directly into your project's Assets folder.
- **Package:** Using the native Unity Package Manager, you can add this library as a package by modifying your `manifest.json` file found at `/ProjectName/Packages/manifest.json` to include it as a dependency. See the example below on how to reference it.

```
{
 "dependencies": {
  ...
  "com.jeffcampbellmakesgames.jtween" : "https://github.com/jeffcampbellmakesgames/unity-jtween.git#release/stable",
  ...
 }
}
```

Step 2 - Add Reference to AssemblyDefinition (optional)

Once imported, a new AssemblyDefinition `JCMG.JTween` will become available that contains the runtime JTween code. If the scripts that you need to be able to interact with JTween, they will need to add `JCMG.JTween` as a dependency. Otherwise if they are not this step can be skipped.

Step 3 - Add JCMG.JTween Namespace to Scripts

In the scripts where you want to use JTWeen, make sure to import its namespace. All user-facing code is available in the `JCMG.JTween` namespace.

```
using JCMG.JTween;
```

Overview

Using JTween should be a straightforward experience if you've ever used similar tween libraries for Unity, despite the complex underlying implementation. The biggest difference which I'll cover in more detail below is that this library de-emphasizes returning any kind of OOP access to the tween itself; for the most part when tweens are created they automaticaly begin playing and will clean themselves up automatically once completed. It is stil possible however to get a reference object to the tween itself in the form of an `ITweenHandle` which allows for more standard user-control over the tween itself; this comes with the responsibility over its lifecycle and requires user action when the tween is no longer needed.

Adding JTweenControl

`JTweenControl` is a global tween manager component which ultimately all tween calls are routed through and managed by. This can be added to an existing scene at edit time by:

- Using the the `Add Component` menu to add it to an existing `GameObject`.
- Using the menu item *Tools->JTween->Add JTweenControl to Scene*. This will add the component to the scene on a new `GameObject` named *JTweenControl* if it does not already exist.

If at runtime `JTweenControl` does not exist, it will be created the first time any type of tween is created.

Creating Tweens

Creating tweens can be done by either using the provided extension methods which offer a variety of specific ways of animating transforms or by making calls directly to `JTweenControl.Instance` which offers all of the tweening methods that are available with all parameters. The example below covers the same tween data, but shows how they vary when created through extension methods or `JTweenControl` directly.

Extension method example

The extension methods for tweens generally focus on a specific type of tween transformation to simplify the API and require less mandatory parameters.

```
// This movement tween will move this transform in local space relative to its parent to the target over 2 seconds
// with a BounceOut ease type and will loop 5 times from the beginning.
transform.MoveLocal(new Vector3(10, 0, 0), 2, EaseType.BounceOut, LoopType.Restart, 5);
```

JTweenControl example

The methods for tweening on `JTweenControl` provide all the possible parameters for a tween transformation. Default values are provided to help simplify calling these methods, but for specific type of transformations (Local vs World) certain parameters like `SpaceType` must be provided to set the correct transformation behavior.

```
// This movement tween will move this transform in local space relative to its parent to the target over 2 seconds
// with a BounceOut ease type and will loop 5 times from the beginning.
JTweenControl.Instance.Move(
    transform,
    transform.localPosition,
    transform.localPosition + new Vector3(10,0,0),
    2,
    SpaceType.Local,
    EaseType.BounceOut,
    LoopType.Restart,
    5);
```

ITweenHandle and Events

By design, most tweens are played automatically once created and cleaned up once completed. This emphasizes a strategy of creating tweens only when needed, avoiding allocating instances of handles to tweens when none may be needed, and avoids requiring by default that users have to manage the lifecycle of a tween. These type of methods will offer `System.Action` callback parameters for when a tween starts and completes to be able to chain relevant functionality to tweens.

However, there are many circumstances where user control over tweens is important, particularly where a user may want to pause, resume, or stop a running tween. This is possible by using the tween method overloads that offer an `out ITweenHandle` parameter. These methods will create tweens that do not play or clean themselves up automatically and require more hands-on managemenent from a user over its lifecycle. Once created, these tweens will remain in the JTween managed data until explicitly recycled. Once a user has recycled an `ITweenHandle`, they should clear any local references to it as it will be reused the next time a user attempts to get another one.

The types of actions a user can execute on an `ITweenHandle` include:

- Add zero or more listeners for when the tween starts.
- Add zero or more listeners for when the tween completes.
- Playing the tween (if this is the first time a user has played the tween, this will invoke any listeners for the started event)
- Pausing the tween.
- Restarting the tween (Rewinds its tween data back to the original values and automatically plays it).
- Rewinding the tween (Rewinds its tween data back to the original values and automatically pauses it. If a tween was already playing, it will be paused and its current state will not be updated until played).
- Stopping the tween (Stops the tween instance and marks it as completed which will invoke any listeners for the completed event).
- Recycle the tween (Once a user has recycled an `ITweenHandle`, any reference to that instance should be cleared).

In addition, the user can actively query for the current state of the tween using the `ITweenHandle` methods:

- `IsPlaying()`
- `IsPaused()`
- `IsCompleted()`

Non-ITweenHandle API Example

API

```
public static void Move(
   this Transform transform,
   Vector3 to,
   float duration,
   EaseType easeType = EaseType.Linear,
   LoopType loopType = LoopType.None,
   int loopCount = 0,
   Action onStart = null,
   Action onComplete = null)
{
 ...
}
```

Example

```
// This tween will play automatically and once completed playing will clean itself up without user
// management.
transform.MoveLocal(new Vector3(10, 0, 0), 2, EaseType.BounceOut, LoopType.Restart, 5, OnTweenStart,
OnTweenComplete;
```

ITweenHandle API Example

API

```
public void Move(
   Transform target,
   Vector3 from,
   Vector3 to,
   float duration,
   out ITweenHandle tweenHandle,
   SpaceType spaceType = SpaceType.World,
   EaseType easeType = EaseType.Linear,
   LoopType loopType = LoopType.None,
   int loopCount = 0)
{
 ...
}
```

Example

```
ITweenHandle tweenHandle;
transform.MoveLocal(new Vector3(10, 0, 0), 2, out tweenHandle, EaseType.BounceOut, LoopType.Restart, 5);
```

Single and Batch Tween APIs

Many API methods exist both as extension methods and on `JTweenControl` for tweening individual transforms. However, where there are many tweens that stop at the same time there is a performance hit for cleaning them up. If starting many tweens on a single frame that have the same duration it may be advantageous to take advantage of the batch APIs on `JTweenControl`. These allow for the creation of many tweens at once whose data is stored in a linear fashion and when completed can be cleaned up much more efficiently. There are some differences to take into account when using these methods, particularly around events and `ITweenHandle`s if used.

Single Tween APIs

- Started and Completed events fire for this tween instance only (1:1 for events per tween)
- An `ITweenHandle` affects only this tween instance.
- Accepts a single transform parameter as well as single parameters for movement, rotation, and scaling.

Batch Tween APIs

- Started and Completed events fire for when the batch of tween instances start and stop/complete (1:X for events to X number of tweens in the batch)
- An `ITweenHandle` reference affects the entire batch of tweens and has the same behavior as it does for a single tween. The batch will start as being paused until played, when paused it will pause every tween in the batch, etc...The batch of data will remain in JTween's managed collections until recycled by a user.
- Accepts an array of transforms as well as arrays for any of the relevant data for movement, rotation, and scaling. Usage is predicated on the arrays having equal length or for sliced versions that the slice is contained within the array's contents; if this is not the case, an assertion will occur.
- Batch methods are available for using an entire array of transforms or only a slice of the array (a linear block of an existing array identified by a start index and length of the array's data from that index that should be used).

Batch API Example Without Slice

```
public void BatchMove(
   Transform[] targets,
   Vector3[] fromArray,
   Vector3[] toArray,
   float duration,
   out ITweenHandle tweenHandle,
   SpaceType spaceType = SpaceType.World,
   EaseType easeType = EaseType.Linear,
   LoopType loopType = LoopType.None,
   int loopCount = 0)
{
 ...
}
```

Batch API Example With Slice

```
public void BatchMoveSlice(
   Transform[] targets,
   Vector3[] fromArray,
   Vector3[] toArray,
   int startIndex,
   int length,
   float duration,
   SpaceType spaceType = SpaceType.World,
   EaseType easeType = EaseType.Linear,
   LoopType loopType = LoopType.None,
   int loopCount = 0,
   Action onStart = null,
   Action onComplete = null)
{
 ...
}
```

Tween Collections

More complex chaining or grouping of tweens can be tedious if attempting to do this manually by using either single or batched listeners for Started, Completed events. For this purpose I have created two tween set collections, `ITweenSet` and `ITweenSequence` which help to make this easier by providing a similiar API to `ITweenHandle`, but allows for managing multiple `ITweenHandle` instances at once. Since an `ITweenHandle` could represent either a single tween or batch of tweens, these collections do not distinguish between them and can handle them both equally well.

NOTE: For tweens that have been set to infinitely loop, these may cause certain undesired behavior such as certain callbacks not being able to occur or blocking progress in the sequence in the case of `ITweenSequence`. Please make sure if you do add inifnitely looping tweens that you are tracking and managing their their ITweenHandles to avoid these scenarios.

ITweenSet

`ITweenSet` is an tween collection that allows a user to add X number of `ITweenHandle` instances and be able to seamlessly execute play, pause, stop, rewind, restart, or recycle actions on all of them at once. In addition, it offers the ability to add listeners for when the `ITweenHandles` are first played and when all of them have completed.

Creating a new `ITweenSet` is as simple as calling `JTweenControl.Instance.NewSet()`.

ITweenSequence

`ITweenSequence` is a tween collection that allows a user to add X number of `ITweenHandle` instances that can be played in sequence. Once the `ITweenSequence` is played, is started event will fire and each `ITweenHandle` will play in the order that they were added. Once all `ITweenHandles` have been played, the completed event will fire. There are a couple of distinctions from `ITweenSet` to take note of:

- Pausing an `ITweenSequence` will pause the currently playing `ITweenHandle` in the sequence if any.
- Restarting an `ITweenSequence` will rewind all `ITweenHandle` instances in the sequence and immediately play the first one.

Creating a new `ITweenSequence` is as simple as calling `JTweenControl.Instance.NewSequence()`.

Thanks for considering contributing to JTween! Read the guidelines below before you submit an issue or create a PR.

Project structure

The project structure is split between several branches

- **master:** This is the stable branch and all releases/packages are generated from this.
- **develop:** This is the primary development branch which all PRs should be made to and is generally considered less-stable. This is occasionally merged into **master** and a new release tag/package is generated from this.
- **releases/stable:** This branch is orphaned and contains only the package contents for JTween. This is updated in sync with tagged releases on **master** and each commit that changes these contents should result in the version in **package.json** being changed.

This structure allows for ease of development and quick testing via **master** or **develop**, but clear isolation and separation between the package distribution via **releases/stable**.

Pull requests

Pull requests should be made to the develop branch. The types of pull requests that are highly desirable are generally:

- Bug fixes
- Feature improvements addressing issues discussed on the GitHub repository. This allows for wider discussion of those issues prior to any implementation and has the potential to help filter features or changes that are undesirable before time is spent working on them.

Performance Issues and Pull Requests

For any issues or pull requests regarding performance improvements, at minimum please include details and screenshots describing the issue in as much detail as possible and if relevant showing a before and after profile. Better yet, if you can provide a reproducible example that can be used to demonstrate the issue it will be much more likely that the issue is addressed quickly.

Style Guide

Language Usage

- Mark closed types as sealed to enable proper devirtualization (see here for more info).
- Avoid LINQ usage for runtime usage except where absolutely possible (`ToList` or `ToArray` for example) and avoid using `ForEach`. Using these methods creates easily avoidable garbage (in newer versions of Unity >= 5.6 this is situational to the Collection or if its being used via an interface, but easy to avoid edge cases by not using at all). Editor usage is another story as performance is not as generally important and non-usage of these can be relaxed.

Layout

There is an `.editorconfig` at the root of the repository that can be used by most IDEs to help ensure these settings are automatically applied.

- **Indentation:** 1 tab = 4 spaces (tab character)
- **Desired width:** 120-130 characters max per line
- **Line Endings:** Unix (LF), with a new-line at the end of each file.
- **White Space:** Trim empty whitespace from the ends of lines.

Naming and Formatting

OBJECT NAME	NOTATION	EXAMPLE
Namespaces	PascalCase	`JCMG.JTween.Editor`
Classes	PascalCase	`SemVersion`
Methods	PascalCase	`ParseVersion`
Method arguments	camelCase	`oldValue`
Properties	PascalCase	`Value`
Public fields	camelCase	`value`
Private fields	_camelCase	`_value`
Constants	All Upper Caps with Snake case	`DEFAULT_VERSION`
Inline variables	camelCase	`value`

Structure

- Follow good encapsulation principles and try to limit exposing fields directly as public; unless necessary everything should be marked as private/protected unless necessary. Where public access to a field is needed, use a public property instead.
- Always order access from most-accessible to least (i.e, `public` to `private`).
- Where classes or methods are not intended for use by a user, mark these as `internal`.
- Order class structure like so:
    - Namespace
        - Internal classes
        - Properties
        - Fields
        - Events
        - Unity Methods
        - Primary Methods
        - Helper Methods
- Lines of code that are generally longer than 120-130 characters should generally be broken out into multiple lines. For example, instead of:

```
public bool SomeMethodWithManyParams(int param1, float param2, List<int> param3, out int param4, out int
param5)...
```

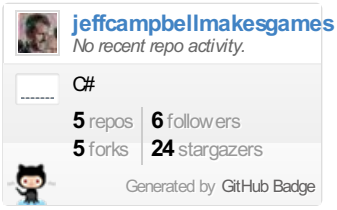
do

```
public bool SomeMethodWithManyParams(
     int param1,
     float param2,
     List<int> param3,
     out int param4,
     out int param5)...
```

Example Formatting

```
using System;
using UnityEngine;

namespace Example
{
    public class Foo : MonoBehavior
    {
        public int SomeValue { get { return _someValue; } }

        [SerializeField]
        private int _someValue;

        private const string WARNING = "Somethings wrong!";

        private void Update()
        {
            // Do work
            Debug.Log(Warning);
        }
    }
}
```

These are users who have contributed in some way to this project, whether that is code, art, design, reporting bugs, or other. The content/format of this will be replaced in the near future using the @all-contributors bot.



Non-legalese summary

- You can freely use JTween in both commercial and non-commercial projects.
- You can modify the code only for your own use and you cannot redistribute modified versions.
- JTween was/is hard work: please respect the copyright.

Definitions

Copyright Holder

Jeff Campbell

You/Your

Means any person who would like to copy, distribute, or modify the Package.

Package

Means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version.

Distribute

Means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization.

Standard Version

Refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder.

Modified Version

Means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder.

License

You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version.

You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers and also include the original readme.txt file. At your discretion, such verbatim copies may or may not include a Compiled form of the Package.

Any use, modification, and distribution of the Standard or Modified Versions is governed by this Artistic License. By using, modifying or distributing the Package, you accept this license. Do not use, modify, or distribute the Package, if you do not accept this license.

This license does not grant you the right to use any trademark, service mark, tradename, or logo of the Copyright Holder.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT

LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.