Welcome to Tween Component

Welcome to Tween Component: an interpolation animation tool aimed to increase your productivity. It allows you to easily create complex tween sequences directly from the editor. No boilerplate, no coding, just plain fun!

- **Easy to use**: just add the Tween Player component to any GameObject, and start animating properties!
- **Artists ready**: allow your artists to easily create animations directly from the Unity editor, with an intuitive UI and simple controls. They can even animate properties from custom scripts without the help of a coder.
- **Flexible**: nest as many Tween Components as you want to create complex animations.
- **Extendible**: Easily create neew Tween Components, with any custom behaviour imaginable.
- **Powerful**: Custom Tweening engine that performs blazingly fast.
- **Coders friendly**: we take a lot of effort making sure the underlying structure is clean and easy to extend. Code aims to be robust and well written.

Contents

Why?

I wanted a tool that allowed teams to:

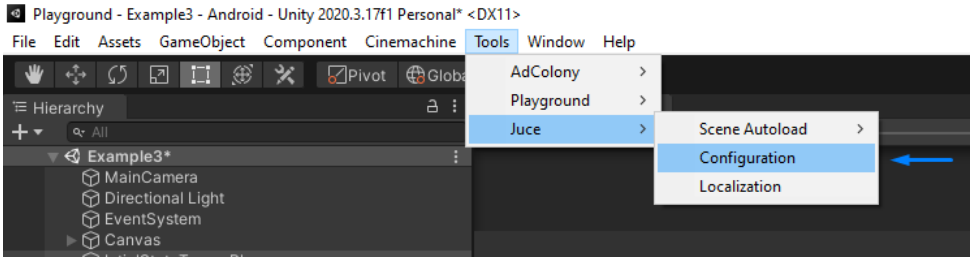
- Add as much juice as they wanted, without compromising the code architecture.
- Give artists a tool that is easy to use, and that does not depend on coders.
- Expand the existing codebase with not much effort.

- Easily bind custom data.

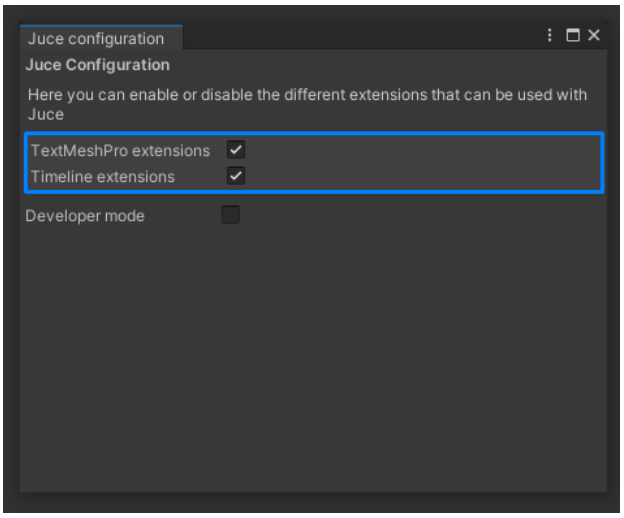
Enabling Extensions (TMPro, Timeline, etc...)

Since Unity is moving towards a very granular approach, we dont want to force our users to have all the dependences installed in your project to start using the tool.

To enanble or disable different extensions, first open on the Unity top bar: ***Tools/Juce/Configuration*** to open the Juce Configuration Window.



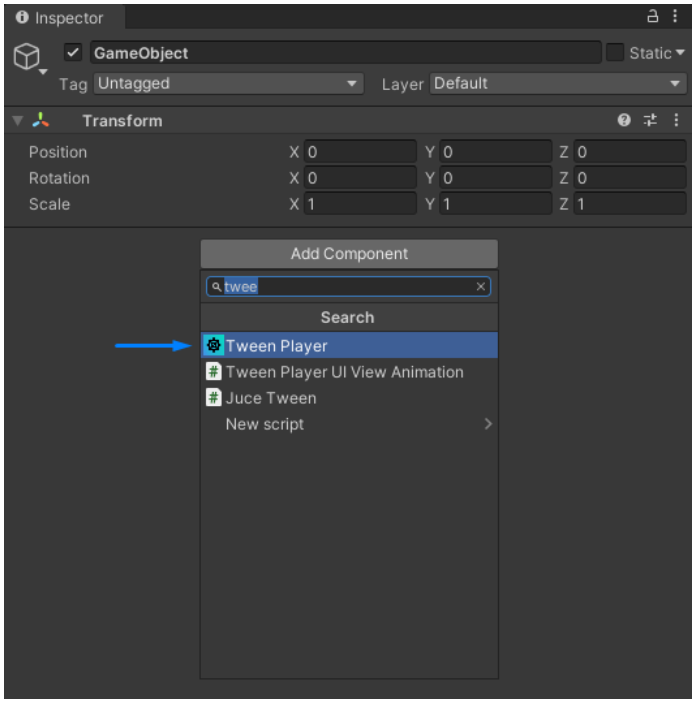
Once the window is opened, you just need to select which extension you want to use in your project. (You may need to wait for Unity to recompile).



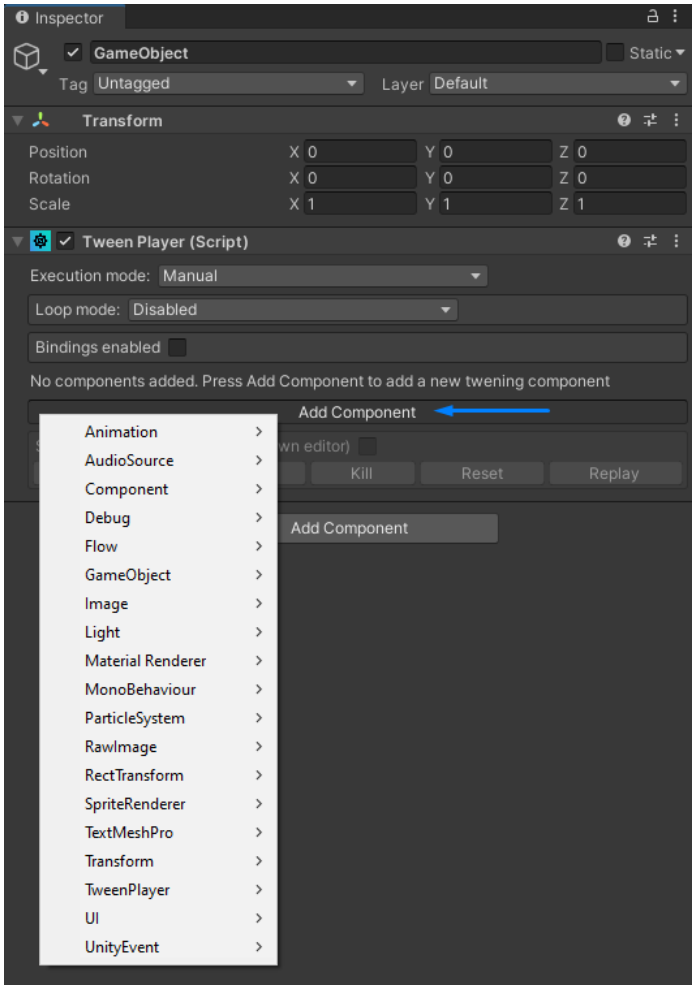
For some Example scenes to work, you will probably need to add, at least, TextMeshPro to your project, and enable the toggle extension.

Basic Usage

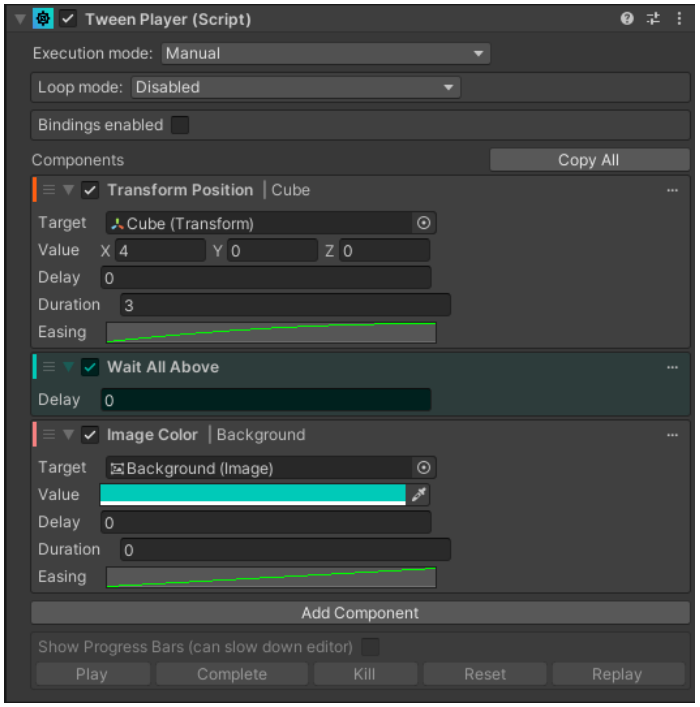
1. Add the Tween Player Component to any GameObject. GameObjects can have more than one Tween Player.



2. Add Tween Components with the Add Component button.

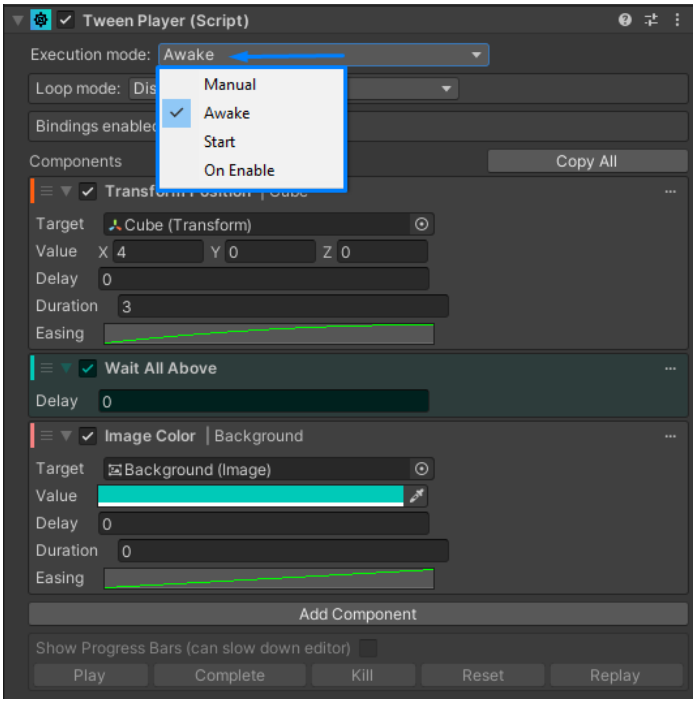


Add as many as you want until the desired animation is reached.



3. Play!

- Directly from the editor:



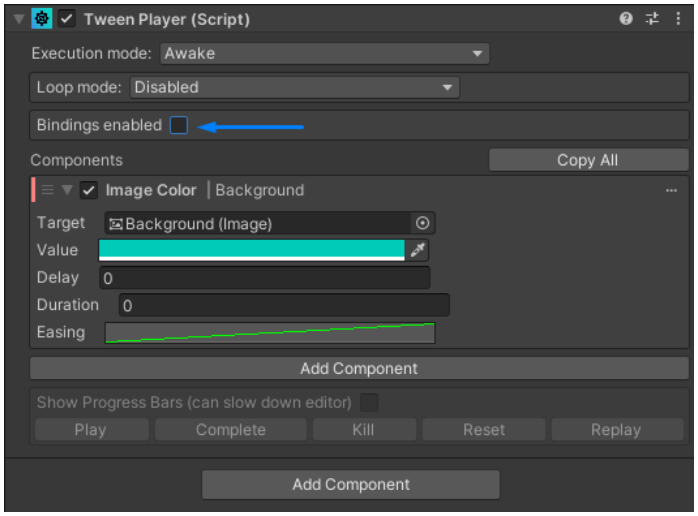
- Through script:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
using Juce.TweenPlayer;

namespace Assets
{
    public class PlayExample : MonoBehaviour
    {
        [SerializeField] private TweenPlayer tweenPlayer = default;

        private void Start()
        {
            // Plays the sequence
            tweenPlayer.Play();

            // Instantly stops the sequence
            tweenPlayer.Kill();

            // Instantly reaches end of sequence
            tweenPlayer.Complete();
        }
    }
}
```

Bindings

Sometimes, you may want to set dynamic values to certain properties of a Tween Component. This is done using Bindings.

1. Every property of every Tween Component can be binded. First of all, you need to enable bindings on a Tween Player component.

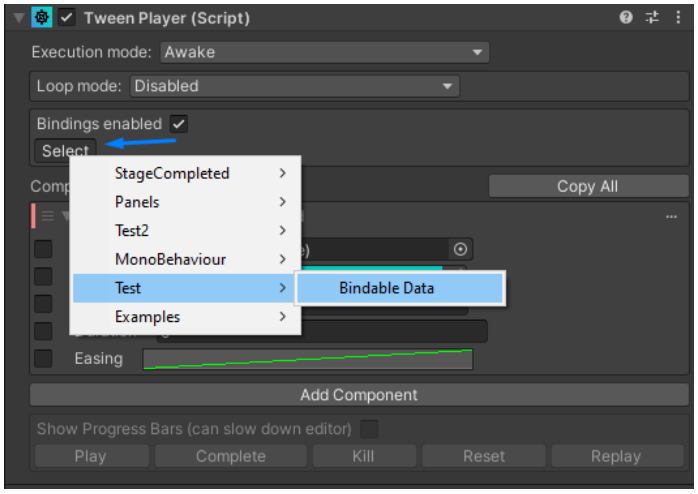


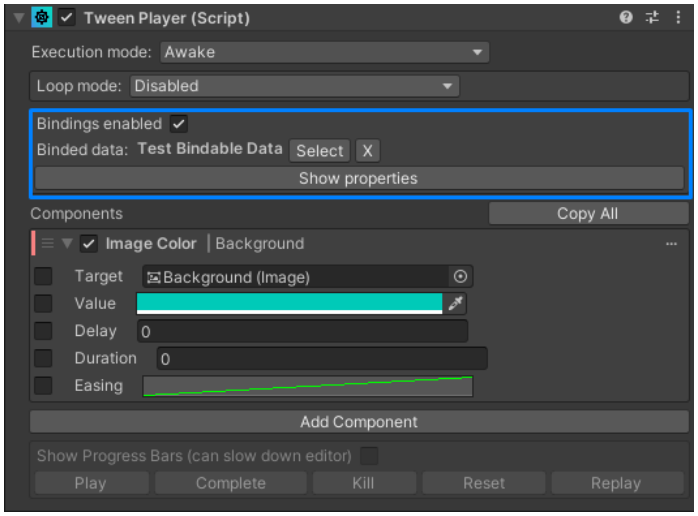
2. Next, you need to define some data to be binded. We define data to bind like this:
   - We define a class that inherits from IBindableData
   - We add the BindableDataAttribute, with the following parameters:
     - Name that the bindable data will have on the Tween Player component
     - Path from which we can find this data from the Tween Player component
     - An identifier, that needs to be unique for all the bindable datas that you have on the project. To avoid unexpected collisions, we recomend to use a random GUID, which can be easily generated, for example, here: https://www.uuidgenerator.net/

       (Having this unique identifier enables the tool to never loose reference to your data, even if you change the class name)

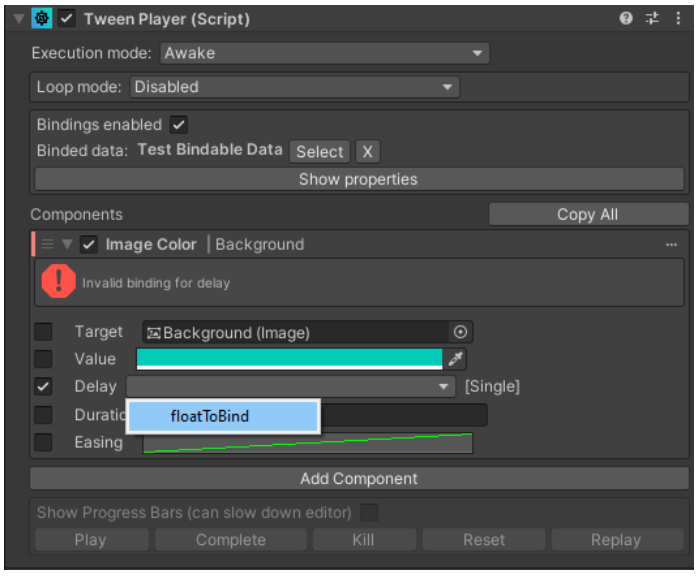
```
using UnityEngine;
using Juce.TweenPlayer.BindableData;

namespace Assets
{
    [BindableData("Test Bindable Data", "Test/Bindable Data", "a8ea3fa2-9e3b-11eb-a8b3-0242ac130003")]
    public class BindableData : IBindableData
    {
        public int intToBind;
        public float floatToBind;
        public string stringToBind;
        public Vector3 vectorToBind;
        public Transform transformToBind;
        // etc...
    }
}
```

Once the new data is created, we should be able to see it through the Tween Player component:





3. When the data is properly set up, you can enable which properties you want to bind from each component (using the toggle found at the left):



   With this, you can define which data goes to which property.

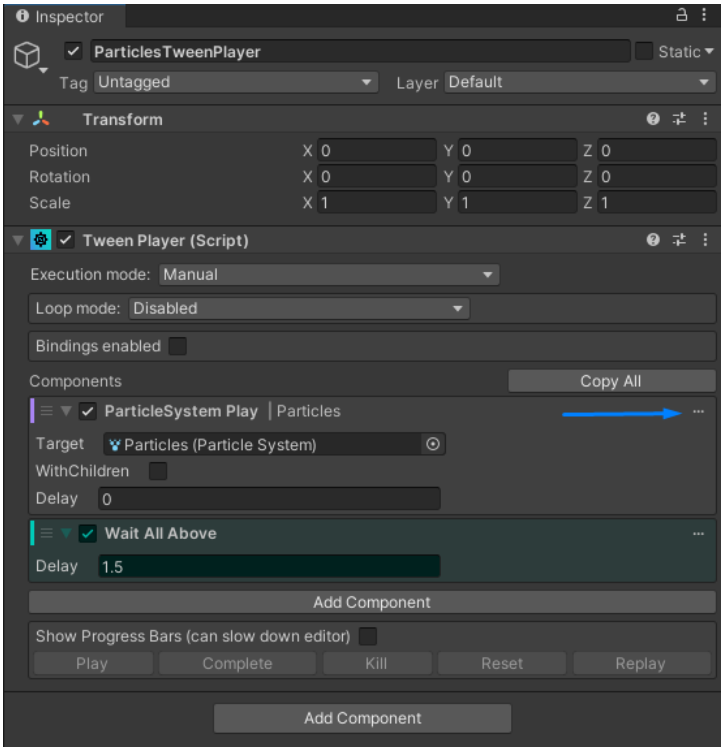
4. Finally, to bind the actual values when you want to play a Tween Player component, you need to pass the actual BindableData class to the Play method, like this:

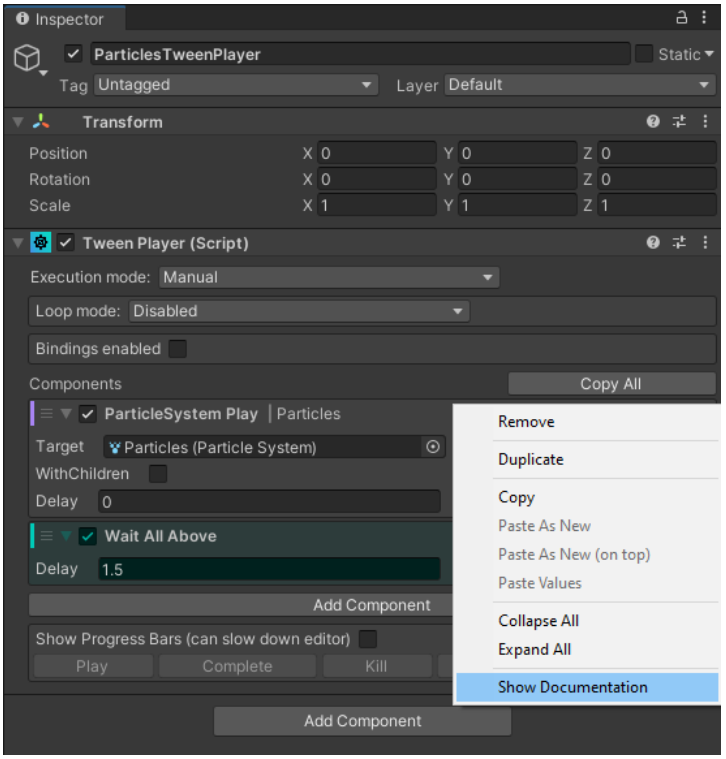
```
using UnityEngine;
using Juce.TweenPlayer;

namespace Assets
{
    public class PlayExample : MonoBehaviour
    {
        [SerializeField] private TweenPlayer tweenPlayer = default;

        private void Start()
        {
            // We create the actual instance of data to play
            BindableData bindableData = new BindableData();
            bindableData.intToBind = 1;
            bindableData.floatToBind = 5.0f;
            bindableData.stringToBind = "Test string";

            // We bind and play the Tween Player
            tweenPlayer.Play(bindableData);
        }
    }
}
```

   Now, your data will be automatically binded to each component using reflection, so you don't need to do anything else!

Tween Components Documentation

You can toggle the components documentation inside the Unity editor by going to a component contextual menu, and selecting Show Documentation.





Want to contribute?

Please follow these steps to get your work merged in.

1. Clone the repo and make a new branch: `$ git checkout https://github.com/Juce-Assets/Juce-TweenPlayer/tree/develop -b [name_of_new_branch] .`
2. Add a feature, fix a bug, or refactor some code :)
3. Update `README.md` contributors, if necessary.
4. Open a Pull Request with a comprehensive description of changes.

Contributors

- Guillem SC - @Guillemsc