Custom Tweening

The Tween class' properties are mostly `{get; internal set;}` so you can't make changes. However, there are some deliberately-exposed properties that you can use for custom tweening.

```
///
<summary>

/// The GameObject reference for GameObject
tweens
///
</summary>

public GameObject? GameObject { get; set;
}



///
<summary>

/// The Action for GameObject
tweening
///
</summary>

public Action<TpTween,float>? GameObjectAction { get; set;
}



///
<summary>

/// Custom user data, example: Used for Bezier curve
evaluation.
///
</summary>
```

```
public object? UserData { get; set;
}


///
<summary>

/// Copy of original user data for
rewinds
///
</summary>

public object? OriginalUserData { get; set;
}


///
<summary>

/// The Renderer for Color
Tweens
///
</summary>

public Renderer? Renderer { get; set;
}


///
<summary>

/// The SpriteRenderer for Color
tweens
///
</summary>

public SpriteRenderer? SpriteRenderer { get; set;
}
```

```
///
<summary>

/// TRUE for GameObject tweens using the
SpriteRenderer
///
</summary>

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

These exposed properties let you create custom tweens that don't use the built-in actions. Care must be taken **NEVER EVER** change any of these once a tween is running.

The field and property values of whatever you use for `UserData` are modified during the custom tween's execution. The `UserData` class' ToString method is used to show information in the Services Inspector and the Tween Monitor windows. You can see how this custom data is used below.

As mentioned earlier, tweens created by one of the Create tween methods launch immediately. However, immediately means the next Update. So a new tween can change these public properties right after creation.

Here's an example of a Component that can be attached to a legacy TextMesh component to tween the character size. Obviously this same approach can be used to tween anything that takes a float value. In essence, this approach is similar to using DOTween's 'Getters' and 'Setters' for specialized tweens.

Follow along: additional comments in CAPS

```
[RequireComponent(typeof(TextMesh))]
public class TextFontSizeTweenExample : MonoBehaviour
{
   private class ActionData
   {
      public float StartValue { get; set; } = 0;   //THE START VALUE
      public float EndValue   { get; set; } = 1;   //THE END VALUE

      public float CurrentValue { get; set; }      //CURRENT VALUE
```

```
      public TextMesh? TextMesh { get; set; } = null;   //TEXTMESH
REFERENCE

      /// <inheritdoc />
      public override string ToString()
      {
          return $"Char Size: Current: {CurrentValue}";
      }
   }



    /// <summary>
    /// Min
    /// </summary>
    [Tooltip("Minimum Character size")]
    public float m_MinimumCharSize = 1f;

    /// <summary>
    /// Max
    /// </summary>
    [Tooltip("Maximum Character size")]
    public float m_MaximumCharSize = 1f;

    /// <summary>
    /// duration
    /// </summary>
    [Tooltip("tween duration")]
    public float m_Duration = 2f;

    void Start()
    {
        var textMesh = GetComponent<TextMesh>();
        if (!textMesh)
            return;

        //GET THE TWEENER SERVICE HANDLE
        var tweener          = TpServiceManager.GetServiceOfType<TpTweener>();
        if (!tweener)
            return;
```

```
        //CREATE AND INIT AN INSTANCE OF OUR CUSTOM DATA
        var ld = new ActionData
                 {
                     StartValue   = m_MinimumCharSize,
                     EndValue     = m_MaximumCharSize,
                     TextMesh     = textMesh,
                     CurrentValue = textMesh.characterSize
                 };

        //Create a custom GameObject tween with our own custom GameObjectAction.
        var id = tweener.CreateGoTween(gameObject,
                                       GameObjectAction,
                                       TpEasingFunction.Ease.CustomEase, //this enum value
causes GameObjectAction to be invoked.
                                       m_Duration,
                                       0,
                                       TpTweener.LoopType.PingPong,
                                       -1);



        var tween = tweener.GetTween(id); //note that this would be incorrect if the tween was
part of a sequence.
        //If we have a sequence ID use GetTweenFromSequence.
        //If it's not known then generally use:
        //   tween = sequenceId != 0 ? GetTweenFromSequence(sequenceId, id) : GetTween(id);

        if (tween == null)
            return;

        //ADD CUSTOM DATA REFERENCES TO THE TWEEN
        //Add our data to the tween, it's sent to the Action below.
        tween.UserData                = ld;
        tween.OriginalUserData        = ld;
        return;

        //This action does exactly the same thing as built in actions
        //but we use the User data items.
```

```
        //The Tweener handles looping, etc.
        //One can hook into the OnRewindOrPingPong to customize this to some extent.
        void GameObjectAction(TpTween t, float progress)
        {
            //unboxing
            if(t.UserData is not ActionData d || d.TextMesh == null)
                return;
            //get start and end values for Lerp
            var sValue = t.Forward ? d.StartValue : d.EndValue;
            var eValue = t.Forward ? d.EndValue : d.StartValue;

            //compute a new value
            var newValue = Mathf.Lerp(sValue, eValue, progress);

            d.CurrentValue = newValue;
            d.TextMesh.characterSize  = newValue;
            //THE TARGET FOR THIS TWEENED VALUE CAN BE ANYTHING THAT TAKES A FLOAT, JUST
CHANGE THE ACTIONDATA CLASS.
        }
    }


}
```

Other examples

See the TilePlusExtras/SimpleDemos/TpTweener/GameObjectExamples project folder.

Future Expansions

Additional built-in tweens for various Unity 'things' will be added over time. But they are really easy to create on your own, as seen here.

Revision #16
Created 2025-08-25 17:13:30 UTC by Vonchor
Updated 2025-09-06 16:03:47 UTC by Vonchor