This is an overview of PlugIns for TMPEffects.

- [AutoParameters](autoparameters.md)

AutoParameters

AutoParameters greatly streamlines the handling of parameters of animations and minimizes boilerplate -- you no longer will have to implement ValidateParameters, SetParameters and GetNewCustomData. If you haven't yet, look at [Creating Animations](#) before this section.

Installation

Using AutoParameters

Once you installed AutoParameters, you can decorate any partial animation class (i.e. one that implements ITMPAnimation) with the [AutoParameters] attribute. This allows you to use the other AutoParameters attributes.

- [AutoParameter] You can decorate any field of your class with this attribute, assuming it is of a [supported type](#).
  You can define whether the parameter is required (false by default), its name as well as any desired amount of aliases.
  For every field decorated with this attribute, a field of the same name will be created in the [custom data object](#).

```
[AutoParameters]
public partial class MyAnimation : TMPAnimation
{
    [AutoParameter("amplitude", "amp"), SerializeField]
    float amp;

    [AutoParameter("color", "colour", "col"), SerializeField]
    Color color;

    [AutoParameter(true, "someOtherValue"), SerializeField]
    int val;
}
```

- [AutoParameterBundle] Alternative to [AutoParameter], to be used with predefined parameter sets. This is at the moment used only for [Waves](#). You may define the prefix used for the wave.

```
[AutoParameters]
public partial class MyAnimation : TMPAnimation
{
    [AutoParameterBundle(""), SerializeField]
    Wave wave;

    [AutoParameterBundle("w2:"), SerializeField]
```

```
    Wave wave2;
}
```

- [AutoParametersStorage] You can decorate exactly one nested, partial type with this attribute. This type will then be used as the [custom data object](#) for the animation. You can add any other fields in here that don't have anything to do with parameters (for example, a RNG). Any initialization unrelated to parameters can be done in the type's default constructor (one without arguments), or in the GetCustomData_Hook (see below).
  ⚠️ If you define a constructor for this type with arguments, you will also have to define an empty constructor.

If you don't decorate any nested type with this attribute, a type called AutoParameterStorage_Generated will be automatically generated and used as data object.

```
[AutoParameters]
public partial class MyAnimation : TMPAnimation
{
    [AutoParametersStorage]
    private partial class Data
    {
        public System.Random rng;
        public Dictionary<int, float> someMapping;

        public Data()
        {
            rng = new System.Random();
            someMapping = new Dictionary<int, float>();
        }
    }
}
```

Hooks

You can hook into each of the generated methods.

- bool ValidateParameters_Hook(IDictionary<string, string> parameters)
  Called at the very beginning of ValidateParameters .
  Rest of validation code is only executed if your method returned true .

- void SetParameters_Hook(object customData, IDictionary<string, string> parameters)
  Called at the very end of SetParameters .
  Is NOT called if passed in parameters dictionary is null.

- void GetNewCustomData_Hook(object customData)
  Called at the very end of GetNewCustomData .
  Receives the custom data object fully populated with all [AutoParameter] .

Full example

The above is all you need to know to use AutoParameters!
This plugin removes all parameter-related boilerplate from your animations and allows you to focus on writing the actual animation logic.
For example, below is the built-in wave animation, if it was written with AutoParameters:

```
[AutoParameters]
public partial class MyAnimation : TMPAnimation
{
    [SerializeField, AutoParameterBundle("")] Wave wave = new
Wave(AnimationCurveUtility.EaseInOutSine(), AnimationCurveUtility.EaseInOutSine(), 0.5f,
0.5f, 1f, 1f, 0.2f);
    [SerializeField, AutoParameter("waveoffset", "waveoff", "woff")] WaveOffsetType
waveOffsetType = WaveOffsetType.XPos;

    public override void Animate(CharData cData, IAnimationContext context)
    {
        Data data = (Data)context.CustomData;

        // Evaluate the wave based on time and offset
        float eval = data.wave.Evaluate(context.AnimatorContext.PassedTime,
GetWaveOffset(cData, context, data.waveOffsetType)).Item1;

        // Move the character up based on the wave evaluation
        cData.SetPosition(cData.InitialPosition + Vector3.up * eval);
    }

    [AutoParametersStorage]
    private partial class Data
    {    }
}
```