PARTICLE SYSTEM MODULES – FAQ

KARL JONES, APRIL 20, 2016

Starting with Unity 5.3, you have full scripting access to the particle system's modules. We noticed these new scripting features can be a bit confusing. Why do we use structs in an unconventional manner?

In this guide, we will answer some of your questions, take a little peek behind the scenes and explain some of our plans to make the process more intuitive in the future.

Accessing Modules

An example

Here is a typical example in which we modify the rate property from the Emission module.

```
using UnityEngine;

public class AccessingAParticleSystemModule : MonoBehaviour
{
  // Use this for initialization
  void Start ()
  {
    // Get the emission module.
    var forceModule = GetComponent<ParticleSystem>().forceOverLifetime;

    // Enable it and set a value
    forceModule.enabled = true;
    forceModule.x = new ParticleSystem.MinMaxCurve(15.0f);
  }
}
```

To anyone familiar with .NET, you may notice that we grab the struct and set its value, but never assign it back to the particle system. How can the particle system ever know about this change, what kind of magic is this!?

It's just an interface

Particle System modules in Unity are entirely contained within the C++ side of the engine. When a call is made to a particle system or one of its modules, it simply calls down to the C++ side.
Internally, particle system modules are properties of a particle system. They are not independent and we never swap the owners or share them between systems. So whilst it's possible in script to grab a module and pass it around in script, that module will always belong to the same system.

To help clarify this, let's take a look at the previous example in detail.
When the system receives a request for the emission module, the engine creates a new EmissionModule struct and passes the owning particle system as its one and only parameter. We do this because the particle system is required in order to access the modules properties.

```
public sealed class ParticleSystem : Component
{
.....

  public EmissionModule emission { get { return new EmissionModule(this); }

....
}
```

```
public partial struct EmissionModule
{
  private ParticleSystem m_ParticleSystem; // Direct access to the particl

  EmissionModule(ParticleSystem particleSystem)
  {
    m_ParticleSystem = particleSystem;
  }

  public MinMaxCurve rate
  {
    set
    {
      // In here we call down to the c++ side and perform what amounts to
      m_ParticleSystem->GetEmissionModule()->SetRate(value);
    }
  }
......
}
```

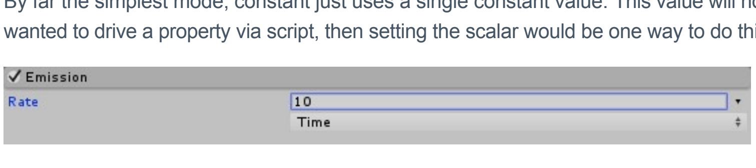
When we set the rate, the variable m_ParticleSystem is used to access the module and set it directly. Therefore we never need to reassign the module to the particle system, because it is always part of it and any changes are applied immediately. So all a module does is call into the system that owns it, the module struct itself is just an interface into the particle systems internals.
Internally, the modules store their respective properties and they also hold state based information, which is why it is not possible for modules to be shared or assigned to different particle systems.
If you do want to copy properties from one system to another, it is advised that you only copy the relevant values and not the entire class, as this results in less data being passed between the C++ side and the C# side.

The MinMaxCurve

A significant number of module properties are driven by our MinMaxCurve class. It is used to describe a change of a value with time. There are 4 possible modes supported; here is a brief guide on how to use each when scripting.

Constant

By far the simplest mode, constant just uses a single constant value. This value will not change over time. If you wanted to drive a property via script, then setting the scalar would be one way to do this.

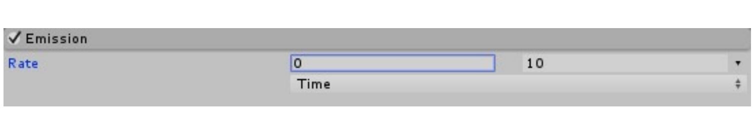


In script, we would access the constant like this:

```
using UnityEngine;

public class MinMaxCurveConstantMode : MonoBehaviour
{
  ParticleSystem myParticleSystem;
  ParticleSystem.EmissionModule emissionModule;

  void Start()
  {
    // Get the system and the emission module.
    myParticleSystem = GetComponent<ParticleSystem>();
    emissionModule = myParticleSystem.emission;

    GetValue();
    SetValue();
  }

  void GetValue()
  {
    print("The constant value is " + emissionModule.rate.constantMax);
  }

  void SetValue()
  {
    emissionModule.rate = new ParticleSystem.MinMaxCurve(10.0f);
  }
}
```

Random between two constants

This mode generates a random value between two constants. Internally, we store the two constants as a key in the min and max curves respectively. We get our value by performing a lerp between the 2 values using a normalised random parameter as our lerp amount.This means that we are almost doing the same amount of work as the random between two curves mode.



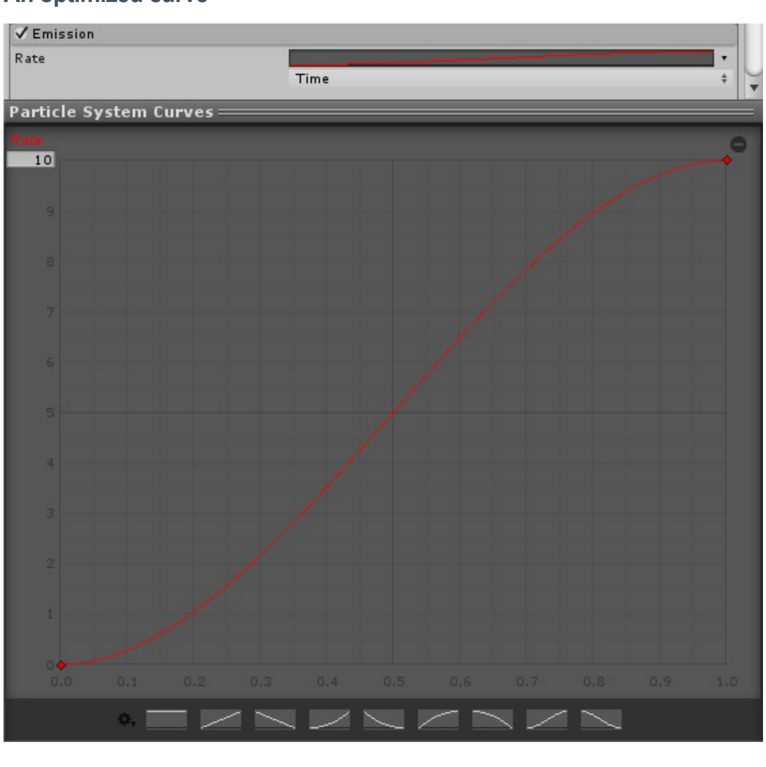
This is how we would access the two constants of a module:

```
using UnityEngine;

public class ParticleModuleExamples : MonoBehaviour
{
  ParticleSystem myParticleSystem;
  ParticleSystem.EmissionModule emissionModule;

  void Start()
  {
    // Get the system and the emission module.
    myParticleSystem = GetComponent<ParticleSystem>();
    emissionModule = myParticleSystem.emission;

    GetRandomConstantValues();
    SetRandomConstantValues();
  }

  void GetRandomConstantValues()
  {
    print(string.Format("The constant values are: min {0} max {1}.", emiss
  }

  void SetRandomConstantValues()
  {
    // Assign the curve to the emission module
    emissionModule.rate =new ParticleSystem.MinMaxCurve(0.0f, 1.0f);
  }
}
```

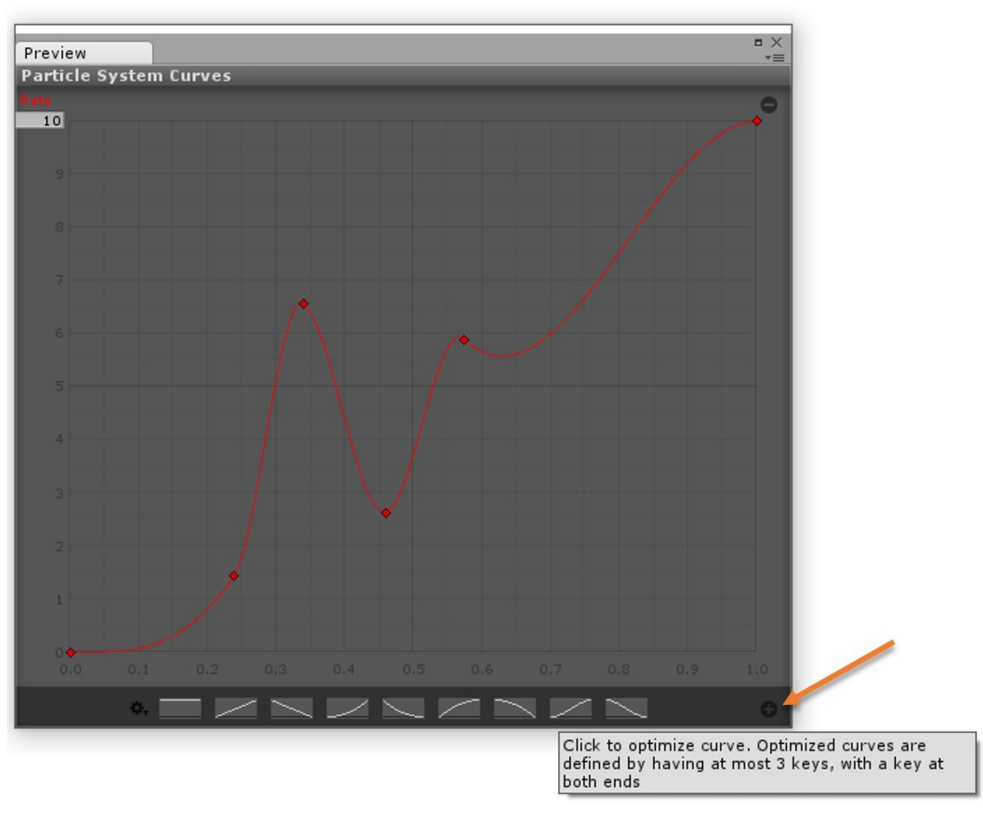
Curve

In this mode, the value of the property will change based on querying a curve. When using curves from the MinMaxCurve in script there are some caveats.
Firstly, if you try to read the curve property when it is in one of the Curve modes, then the you'll get the following error message: *"Reading particle curves from script is unsupported unless they are in constant mode"*.
Due to the way the curves are compressed in the engine, it is not possible to get the MinMaxCurve unless it is using one of the 2 constant modes.We know this isn't great, read on our plans to improve it. The reason for this is that internally, we don't actually just store an AnimationCurve, but select one of two paths. If the curve is simple (no more than 3 keys with one at each end), then we use an optimized polynomial curve, which provides improved performance. We then fallback to using the standard non-optimized curve if it is more complex. In the Inspector, an unoptimized curve will have a small icon at the bottom right which will offer to optimize the curve for you.

An optimized curve



An unoptimized curve

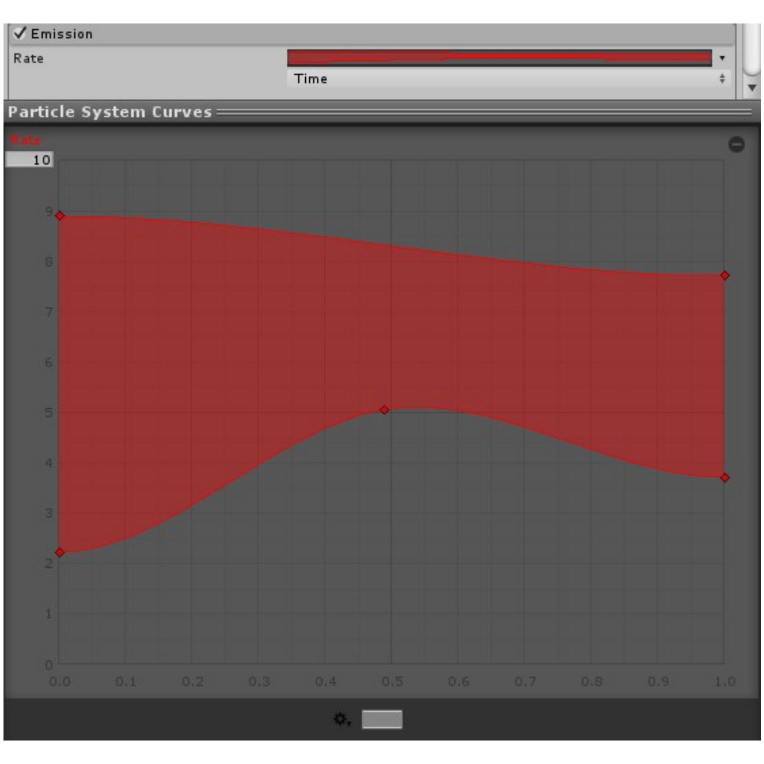


While it may not be possible to get the curve from a module in script, it is possible to work around this by storing your own curve and applying this to the module when required, like this:

```
using UnityEngine;

public class MinMaxCurveCurveMode : MonoBehaviour
{
  ParticleSystem myParticleSystem;
  ParticleSystem.EmissionModule emissionModule;

  // We can "scale" the curve with this value. It gets multiplied by the c
  public float scalar = 1.0f;

  AnimationCurve ourCurve;

  void Start()
  {
    // Get the system and the emission module.
    myParticleSystem = GetComponent<ParticleSystem>();
    emissionModule = myParticleSystem.emission;

    // A simple linear curve.
    ourCurve = new AnimationCurve();
    ourCurve.AddKey(0.0f, 0.0f);
    ourCurve.AddKey(1.0f, 1.0f);

    // Apply the curve
    emissionModule.rate = new ParticleSystem.MinMaxCurve(scalar, ourCurve)

    // In 5 seconds we will modify the curve.
    Invoke("ModifyCurve", 5.0f);
  }

  void ModifyCurve()
  {
    // Add a key to the current curve.
    ourCurve.AddKey(0.5f, 0.0f);

   // Apply the changed curve
   emissionModule.rate = new ParticleSystem.MinMaxCurve(scalar, ourCurve);
  }
}
```

Random between 2 curves.

This mode generates random values from in between the min and a max curves, using time to determine where on the x axis to sample. The shaded area represents the potential values. This mode is similar to the curve mode in that it is not possible to access the curves from script and that we also use optimized polynomial curves(when possible). In order to benefit from this, both curves must be optimizable, that is contain no more than 3 keys and have one at each end. Like in curve mode, it is possible to tell if the curves are optimized by examining the bottom right of the editor window.



This example is very similar to the curve, however, we now also set the minimum curve as well.

```
using UnityEngine;

public class MinMaxCurveRandom2CurvesMode : MonoBehaviour
{
  ParticleSystem myParticleSystem;
  ParticleSystem.EmissionModule emissionModule;

  AnimationCurve ourCurveMin;
  AnimationCurve ourCurveMax;

  // We can "scale" the curves with this value. It gets multiplied by the
  public float scalar = 1.0f;

  void Start()
  {
    // Get the system and the emission module.
    myParticleSystem = GetComponent<ParticleSystem>();
    emissionModule = myParticleSystem.emission;

    // A horizontal straight line at value 1
    ourCurveMin = new AnimationCurve();
    ourCurveMin.AddKey(0.0f, 1.0f);
    ourCurveMin.AddKey(1.0f, 1.0f);

    // A horizontal straight line at value 0.5
    ourCurveMax = new AnimationCurve();
    ourCurveMax.AddKey(0.0f, 0.5f);
    ourCurveMax.AddKey(1.0f, 0.5f);

    // Apply the curves
    emissionModule.rate = new ParticleSystem.MinMaxCurve(scalar, ourCurveM

    // In 5 seconds we will modify the curve.
    Invoke("ModifyCurve", 5.0f);
  }

  void ModifyCurve()
  {
    // Create a "pinch" point.
    ourCurveMin.AddKey(0.5f, 0.7f);
    ourCurveMax.AddKey(0.5f, 0.6f);

    // Apply the changed curve
    emissionModule.rate = new ParticleSystem.MinMaxCurve(scalar, ourCurveM
  }
}
```

Performance

We did some simple performance comparisons to see how these different modes compare. These samples were taken before our recent SIMD optimizations, which should provide a significant performance boost. In our specific test scene, we got these results:

Easing the pain

Reading curves from the MinMaxCurve class

We know that you really ought to be able to read particle system curves from script, regardless of what mode they are in. We are actively working on removing this limitation, so you can read/modify all those lovely curves in your scripts. Its also currently not possible to query the curve to check if the mode is using a curve, without an error being thrown. That's also going to be fixed!

Changing modules from structs to classes

We are currently prototyping a change to move all the structs to classes. Functionally they will behave the same, however by using a reference type, it should be clearer that the module belongs to a system. It will also allow for setting/getting values without having to hold a temporary variable. However, this will mean we allocate them once in the constructor, which will generate garbage, but only at initialization time.

For example:

```
var em = ps.emission;
em.enabled = true;

// Could be written as

ps.emission.enabled = true;
```

Finally

We hope this guide has been of some use to you, please head over to this forum post to grab the examples and comment/discuss.