Introduction

Welcome to 2D Trail Effect Wiki. This documentation will help you learn how to use the asset.

2D Trail Effect is a Unity tool that allows you to generate trails from game objects. It is easy to use, and comes with many customisations.

This wiki contains all the features of the free version and pro version. Pro version exclusive features will be marked as PRO ONLY FEATURE or marked by a * in front.

You can download the free version here or in the asset store. If you find this tool useful and would like to have the full set of features, consider buying the Pro version.

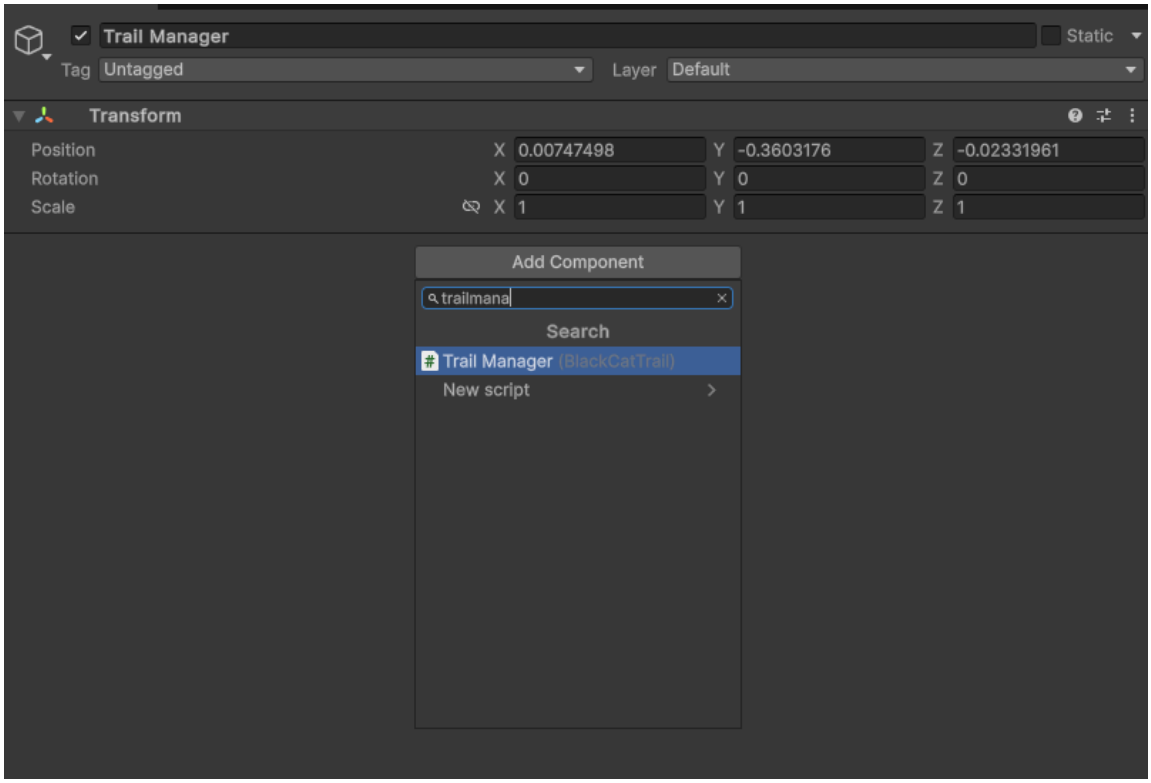
As a start, why don't we visit this page and learn how to set this tool up first?

***Important: Please avoid touching anything in my Resources folder unless you are just moving them to another Resources folder, thank you <(_ _)>.**

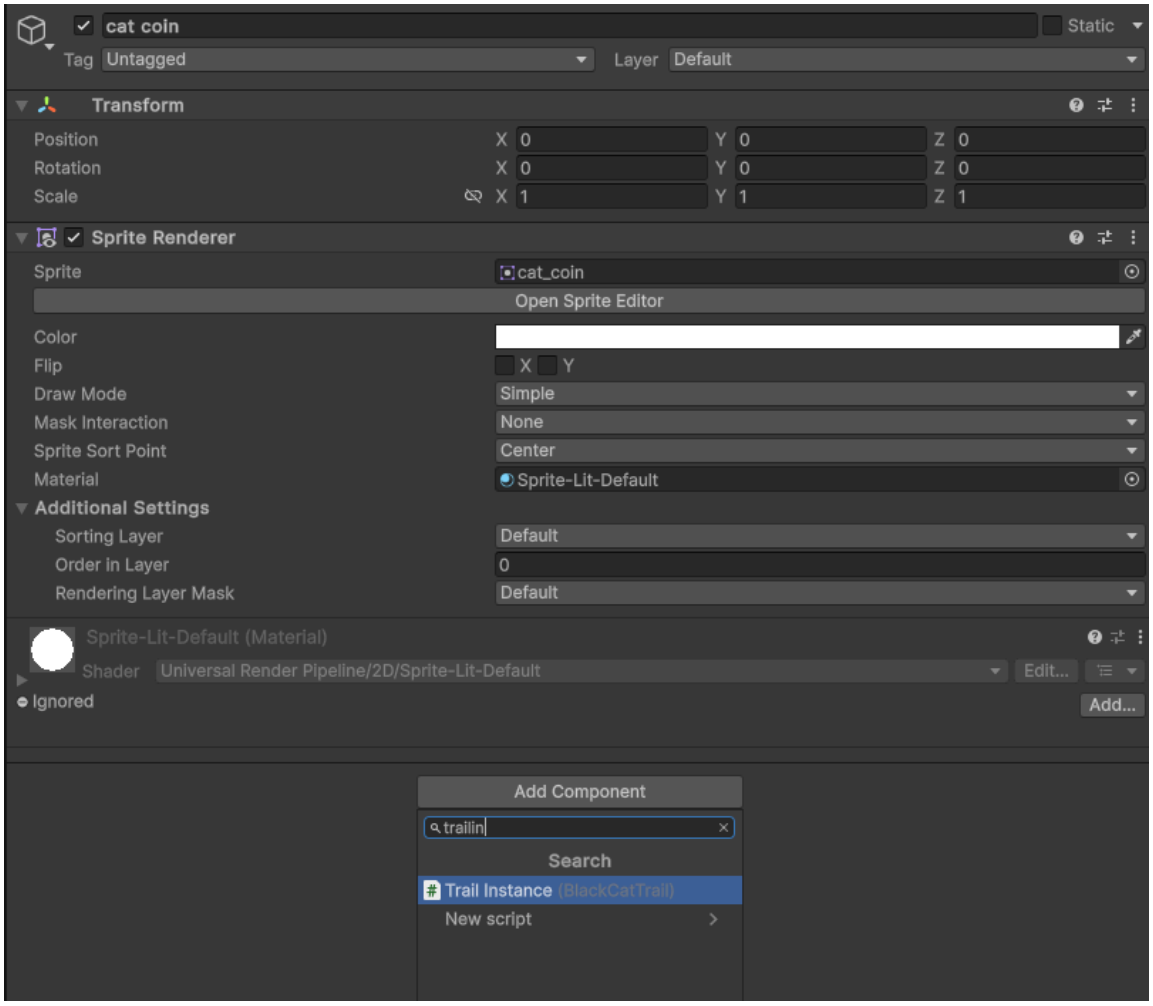
Set Up

Before you can use the trails, you need to set some things up.

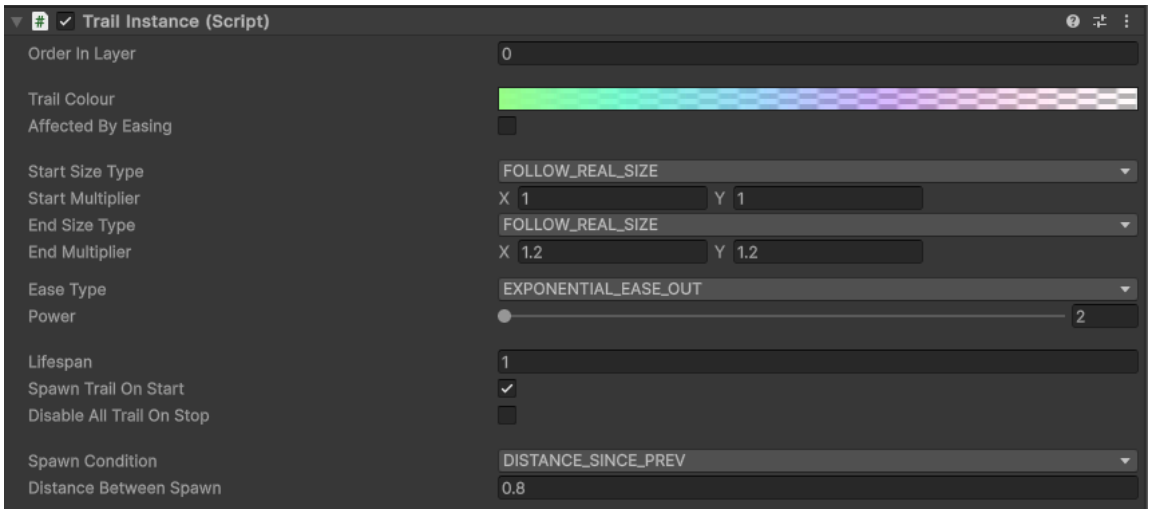
1. Create an empty game object, name it whatever you want, and attach the “Trail Manager” script.



2. On the game object you want to spawn trails of, attach the “Trail Instance” script.



3. Just freely customise the trail!



Basic Usage

Start A Trail

At a place of a script where you want to start a trail, add the following line:

```
BlackCatTrail.TrailManager.Instance.StartTrail(gameObject);
```

***PRO ONLY FEATURE** Or this line:

```
gameObject.StartTrail();
```

“gameObject” is literally the game object your script is attached to. If you want to start the trail on another game object, simply pass that game object to the bracket as the parameter instead.

Note that starting a new trail on a game object that is already spawning trails will immediately stop that trail and start the new trail.

Stop A Trail

At a place of a script where you want to stop a trail, add the following line:

```
BlackCatTrail.TrailManager.Instance.StopTrail(gameObject);
```

***PRO ONLY FEATURE** Or this line:

```
gameObject.StopTrail();
```

Just like StartTrail, “gameObject” is the game object your script is attached to. If you want to stop the trail of another game object, simply pass that game object to the bracket as the parameter instead.

Passing a game object that is not spawning any trails will not give you any errors.

Check If A GameObject Is Spawning A Trail

If you want to check if a game object is spawning a trail, use this line:

```
BlackCatTrail.TrailManager.Instance.IsObjectSpawningTrails(gameObject);
```

***PRO ONLY FEATURE** Or this line:

```
gameObject.gameObject.IsSpawningTrail();
```

Trail Manager

The Trail Manager is the only thing you need to set up, and is the one crucial thing that makes this work. There are 3 things you can adjust on the manager.

Auto Clean Up

Since this tool uses object-pooling, pooling many objects is unavoidable. Luckily, the manager will clear things up automatically. When this is toggled on, the manager will automatically destroy the pooled trail objects if too much time has passed since the last trail has stopped.

Clean Up After Seconds

The time you want the manager to automatically clean up pooled trail objects since the last trail ended in seconds. The manager will clear up things after 10 seconds by default.

Number Of Trail Remains

Auto cleanup is amazing, but surely you might want it to leave some alone. This will be the number of pooled trail objects you want to keep after cleanup.

Manual Clean Up

If you really can't stand the sea of pooled trail objects, you can call this function to destroy them manually:

```
BlackCatTrail.TrailManager.Instance.DestroyTrail(int);
```

“int” is an integer that represents the number of pooled trail objects you wish to destroy.

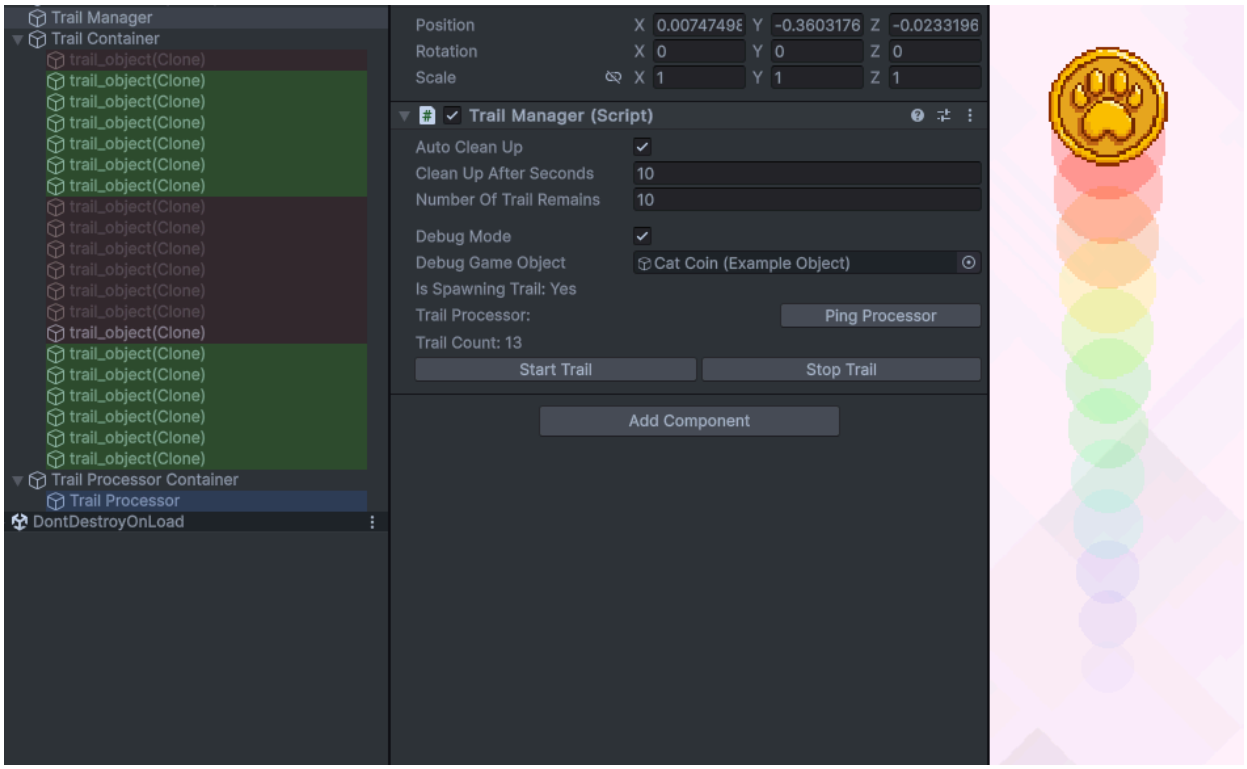
Debugging

At the bottom, you will see a **Debug Mode** toggle. If you turn it on, it will be activated when you enter Play Mode.

When debugging is activated, you can drag a game object in your scene into the Debug **Game Object** field. It will then display whether the game object is spawning a trail, which processor is responsible for it (press **Ping Processor** to ping it in the hierarchy) and the amount of active trail object of the trail.

There are also 2 buttons **Start Trail** and **Stop Trail** so you can manually start and stop them in debug mode.

In the meantime, in the hierarchy, the processor will be highlighted by blue, the pooled trail objects that were used by the trail of the debug game object and were not yet used by other trails will be highlighted by red, and the active trail objects of the trail of the debug game object will be highlighted by green.



If the debugging information is not being updated, simply hover your mouse there and it will update immediately.

Please avoid turning debug mode on when you don't need it.

Trail Instance

This is where you customise your trails. Each field will be explained below.

Profile (PRO ONLY FEATURE)

Profile allows you to save a trail instance and load it on other trail instances, saving you the time and energy to copy and paste the values over and over again.

- **Create a profile:** Give it an identifier (aka a name) on the Profile Identifier field, and click Create.

- **Load a profile:** On the trail instance, on the Profile field, select the profile you want to load, then simply click Load.
- **Delete an existing profile:** Select the profile on the Profile field, then click Delete.
- **Rename an existing profile:** Select the profile on the Profile field, type the new name into the Profile Identifier field, then click Rename.

When creating a profile, if there is already a profile with the same identifier, that profile will be overridden by the new one, and every game object and prefab that has the trail instance conponent and the prodile identifier is the same as the new one created will all be updated.

Switch Profile At Runtime

To change the profile of a trail instance, call this function:

```
gameObject.GetTrailInstance().SetProfile(string);
```

Or if you are already holding the reference to the trail instance, just call SetProfile(string) on it. "string" is the identifier of the profile, which is the **Profile Identifier** field when you create a new profile. If the profile is not found using the identifier, it throws an error.

Order In Layer

As straightforward as it is, controls the **Order In Layer** field on the sprite renderer component.

Trail Sprite (PRO ONLY FEATURE)

If you want your trail to have different sprites instead of just using the game object's sprite, you can set this field to **SPECIFIC**.

When you do, you will see 2 fields: **Sprites** and **Appear Order**.

Sprites

This is where you specify the sprites you want to use. If you haven't provided any sprites, then the sprite of the game object will be used instead.

Appear Order

You can specify whether you want the sprite to be used in the order you provided (**IN_ORDER**), or randomly picked (**RANDOM**).

Trail Colour

You can control the colour gradient of the trail over time. If **Affected By Easing** is toggled on, then the easing or the animation curve **(PRO ONLY FEATURE)** used to smooth the size of the trail will also be applied to the colour.

And yes, this asset is developed using the URP render pipeline, so blooming colour is supported.

*The colour will normally be tinted into the sprite colour. If you want the colour to override the colour of the sprite completely, toggle **Override Trail Sprite** on.

Trail Size

You can specify whether you want the size of the trail to be following the world scale of the game object (**FOLLOW_REAL_SIZE**) or to be the fixed scale you specify (**FIXED_SIZE**). Specify the type and the values for the trail's starting and ending sizes.

FOLLOW_REAL_SIZE: You can specify the multiplier of the scale in **Start Multiplier** or **End Multiplier** fields. The x and y scale multiplier can be adjusted separately.

FIXED_SIZE: You can provide the fixed scale in **Start Size** and **End Size** fields.

Scale Over Time

In order to make the size changes look smoother, you can choose from using **CURVE (PRO ONLY FEATURE)** or **EASE** in the field **Scale Over Time Type**. (For free version, this field doesn't exist and you are only able to use EASE)

Easing

There are 4 types of easings built in: **Exponential**, **Sine**, **Circle** and **Elastic**. You can choose an easing type in the **Ease Type** field.

If **Exponential** easing is selected, you can adjust the exponent within a range of 2~5.

Feel free to use the easing equations in your own project if you need them. They are in the **BlackCatTrail.Equations** class.

Animation Curve (PRO ONLY FEATURE)

If you want more customisations than just built-in easing equations, you can adjust the animation curve however you want in the **Scale Curve** field.

Lifespan

You can adjust the lifespan aka the existing time of a trail object in seconds in the **Lifespan** field.

To immediately spawn a trail object after a trail started, toggle **Spawn Trail On** Start on.

If you want the trail to immediately disappear after it is stopped, toggle **Disable All Trail On Stop** on.

Auto Start And Stop (PRO ONLY FEATURE) (Experimental)

You can specify the condition for the game object to automatically start and stop a trail.

Auto Start

The current built-in conditions for auto-starting a trail is when the speed of the game object is fast enough.

To enable auto-start by speed, select **ABOVE_WORLD_SPEED** or **ABOVE_LOCAL_SPEED** in the **Auto Start Condition** field.

Those 2 works basically the same. However, world speed is the speed relative to the world, where the local speed is the speed relative to the rotation of the game object.

To make this work, you need to move the game object using the **Rigidbody2D** by **linear velocity**. The **Rigidbody2D** needs to be **dynamic** and **simulated** turned on.

Auto Stop

You can choose from 3 types of built-in conditions:
AUTO_START_CONDITION_NOT_MET, **AFTER_TIME** or **AFTER_DISTANCE**.

AUTO_START_CONDITION_NOT_MET: The trail stops immediately as soon as the condition to auto-start a trail is no longer met. For instance, if the trail starts when the game object is moving, then the trail will stop as soon as the game object stops moving.

AFTER_TIME: The trail stops after a specific time has passed since the trail has started.

AFTER_DISTANCE: The trail stops after the game object has moved for a specific distance.

Custom Condition

If none of the built-in conditions fit you, you can write your own conditions.

For instance, if you want the trail to start when an int **buff_count** is above 3:

```
gameObject.GetTrailInstance().SetCustomAutoStart(() => buff_count > 3);
```

Then if you want the trail to stop when "buff_count" has dropped back to 0:

```
gameObject.GetTrailInstance().SetCustomAutoStop(() => buff_count == 0);
```

Both of these methods should be written in the **Awake** function in a script that is attached to that game object.

Please note that the custom conditions currently will not be changed after profile changes. If you want to change the custom conditions on profile change, please manually set them using code.

Spawn Condition

This is the condition to spawn each trail object when a trail is active. There are 3 conditions: **TIME**, **DISTANCE_SINCE_PREV** AND **DISTANCE_MOVED**.

TIME: Each trail object will be spawned with a specific time interval in between.

DISTANCE_SINCE_PREV: A trail object will spawn as soon as the game object is far away enough from the previous position that spawned a trail object.

DISTANCE_MOVED: A trail obejct will spawn after the game object has moved in total for a specific distance regardless of its current position.

Trail Physical Collision (PRO ONLY FEATURE)

If you want to detect collisions with the trails, you can toggle **Enable Trail** Collision on. You will then see 2 more fields: **Collider Size Multiplier** and **Allow Collision Period**.

Collision Size

To adjust the size of the collider box of the trail objects, you can provide a multiplier in the **Collider Size Multiplier** field. The x and y scale multiplier can be adjusted separately. Note that the collider of the trail object is a box collider.

Collidable Time Period

If you want the trail to only be collidable within a time period, you can adjust the slider in the **Allow Collision Period** field. It is a double slider, so you can adjust the time to enable and disable collision. The maximum time will be the lifespan of the trail.

Code Usage

Let's say in OnTriggerEnter2D, a collision is detected. In order to check if the collided game object is a trail object, you can write this:

```
private void OnTriggerEnter2D(Collider2D collision)
{
```

```
    if (collision.gameObject.IsTrailObject())
    {
        //your actions...
    }
}
```

Additionally, let's say only the trail of a specific game object should take effect, write this instead:

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.GetTrailSource() ==
the_specific_gameobject)
    {
        //your actions...
    }
}
```

Avoid using IsTrailObject() and GetTrailSource() together since this will cause a double TryGetComponent. Most of the time, you will know that the game object is not a trail object if GetTrailSource() returns null.

Extension Methods (PRO ONLY FEATURE)

I added some extension methods so you ~~and I~~ can write shorter code.

Game Object

GetTrailInstance()

This returns the trail instance attached to the game object.

Instead of writing

```
gameObject.GetComponent<TrailInstance>();
```

You can just write

```
gameObject.GetTrailInstance();
```

StartTrail()

This will start a trail from the game object.

Originally, you need to write

```
BlackCatTrail.TrailManager.Instance.StartTrail(gameObject);
```

and pass the game object to the method.

Now, just write

```
gameObject.StartTrail();
```

StopTrail()

This will stop the trail of the game object.

Originally, you need to write

```
BlackCatTrail.TrailManager.Instance.StopTrail(gameObject);
```

and pass the game object to the method.

Now, just write

```
gameObject.StopTrail();
```

IsSpawningTrail()

This tells you if the game object is spawning a trail.

Instead of writing

```
BlackCatTrail.TrailManager.Instance.IsObjectSpawningTrails(gameObject);
```

and pass the game object to the method,

you can just write

```
gameObject.IsSpawningTrail();
```

Or even, if you hold the reference to the trail instance, you can just write

```
TrailInstance instance = gameObject.GetTrailInstance();
//...
if (instance.IsSpawningTrail)
{
    //Your Actions...
}
```

This approach actually improves performance as well as the logic is different.

IsTrailObject()

When you want to check if a game object is a trail object, instead of writing

```
the_gameobject.TryGetComponent(out var TrailObject);
```

Just write

```
the_gameobject.IsTrailObject();
```

GetTrailSource()

This is when you want to get the game object that is spawning the trail object, you can write

```
GameObject trailSource = the_trailobject.GetTrailSource();
```

If this method returns a null, that means either the trail source is destroyed or disabled, or the game object you are calling this method from is actually not a trail object.

Avoid using IsTrailObject() and GetTrailSource() together since this will cause a double TryGetComponent. Most of the time, you will know that the game object is not a trail object if GetTrailSource() returns null.

Gradient

If you want to copy the colour keys and the alpha keys from one gradient to another, you can call this method:

```
destination_gradient.CopyKeys(source_gradient);
```

Animation Curve

If you want to copy the key frames from one animation curve to another, you can call this method:

```
destination_curve.CopyKeys(source_curve);
```

Events (PRO ONLY FEATURE)

I made some events that you can subscribe to.

onProfileChanged

This event will invoke when the profile of a trail instance is changed. This event returns the trail instance whose profile has changed.

For instance, if you find the profile is not switching correctly, you can subscribe to the event and see what is going on:

```
TrailEvents.onProfileChanged += OnProfileChanged;

private void OnProfileChanged(TrailInstance instance)
{
    Debug.Log(instance.Profile.Identifier);
}
```

onTrailStarted

This event will invoke when a trail starts. This event returns the game object that started the trail.

For instance, if you want to play a sound when a trail starts:

```
TrailEvents.onTrailStarted += OnTrailStarted;

private void OnTrailStarted(GameObject obj)
{
    AudioManager.Play("trail_start_sound");
}
```

onTrailStopped

This event will invoke when a trail stops. This event returns the game object whose trail has stopped.

For instance, if you want to change the profile of a trail instance after its trail has stopped:

```
TrailEvents.onTrailStopped += OnTrailStopped;

private void OnTrailStopped(GameObject obj)
{
    if (gameObject == obj)
    {
        gameObject.GetTrailInstance().SetProfile("Another Trail");
    }
}
```

onTrailObjectInitialised

This event will invoke after a trail object is initialised. This event returns the TrailObject component of the trail object and the trail processor responsible for spawning this trail object.

For instance, if you want to change the material of the trail object spawned by a specific game object after it is initialised, you can do this:

```
TrailEvents.onTrailObjectInitialised += OnTrailStarted;

private void OnTrailObjectInitialised(TrailObject trailObject,
TrailProcessor processor)
{
    if (trailObject.gameObject.GetTrailSource() == the_gameobject)
    {
        trailObject.gameObject.GetComponent<SpriteRenderer>().material =
the_material;
    }
}
```

Or

```
TrailEvents.onTrailObjectInitialised += OnTrailStarted;

private void OnTrailObjectInitialised(TrailObject trailObject,
TrailProcessor processor)
{
    if (processor.TrailInstance.gameObject == the_gameobject)
    {
        trailObject.gameObject.GetComponent<SpriteRenderer>().material =
the_material;
    }
}
```

onTrailObjectDisabled

This event will invoke when a trail object is disabled due to expiration or early deactivation (by toggling Disable All Trail On Stop on). This event returns the TrailObject component of the trail object and the trail processor responsible for spawning this trail object.

For instance, if you want to spawn a explosion at the trail object after it disappears if the trail is spawned by a specific game object, you can write:

```
TrailEvents.onTrailObjectDisabled += OnTrailObjectDisabled;
```

```
private void OnTrailObjectDisabled(TrailObject trailObject,
TrailProcessor processor)
{
    if (processor.TrailInstance.gameObject == the_gameobject)
    {
        Vector2 position = trailObject.transform.position;
        SpawnExplosion(position);
    }
}
```

onWillDestroyTrailObject

This event will invoke when a pooled trail object is going to be destroyed. This event returns the TrailObject component of the trail object and the trail processor responsible for spawning this trail object.

For instance, if you want to keep track of how many pooled trail object has been destroyed automatically by the trail manager, which I have no idea why you would ever need to do this, then you can do:

```
TrailEvents.onWillDestroyTrailObject +=
OnWillDestroyTrailObject;

private void OnWillDestroyTrailObject(TrailObject trailObject,
TrailProcessor processor)
{
    if (_Counting == null)
    {
        _Counting = CountDestroyedTrail();
        StartCoroutine(_Counting);
    }
    count++;
}

private IEnumerator CountDestroyedTrail()
{
    yield return null;
    Debug.Log(count);
```

```
}
```

Possible Problems

Gap Inconsistency

Due to frame-rate inconsistency which causes the delta time to be imprecise, inconsistent gaps between each trail object are possible. I implemented some calculations to calculate the spawn positions as precisely as possible so that you will likely not notice it, but I can't guarantee the inconsistent gaps will never appear.

Auto Stop On Destroy

This tool can automatically stop trails if the game objects are destroyed or disabled. However, I cannot guarantee that it always stops before object destruction. If you get into cases where it doesn't, simply stop the trails before destroying the game objects. But in general, it should work 99.9% of the time.

Game Scene Maximisation Failure

Unity throws an error when I try to maximise the game scene after modifying the animation curve. This is a Unity bug as far as I know, and it shouldn't affect the game. If you don't want to see this, the only work around currently is to **1)** restart the editor everytime which probably no one would do, or **2)** click on **Window/Layouts/Reset All Layouts** every time after you edit the animation curve. Otherwise, just ignore it. ***This problem should be fixed on version 6000.2.0a2 according to the Unity team.***