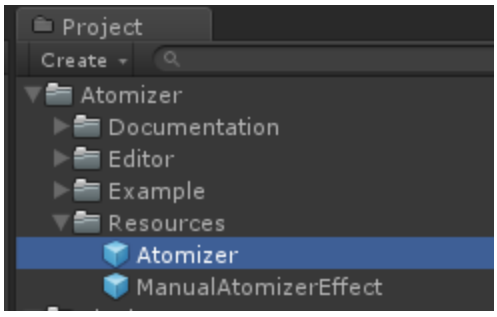
Atomizer User Guide

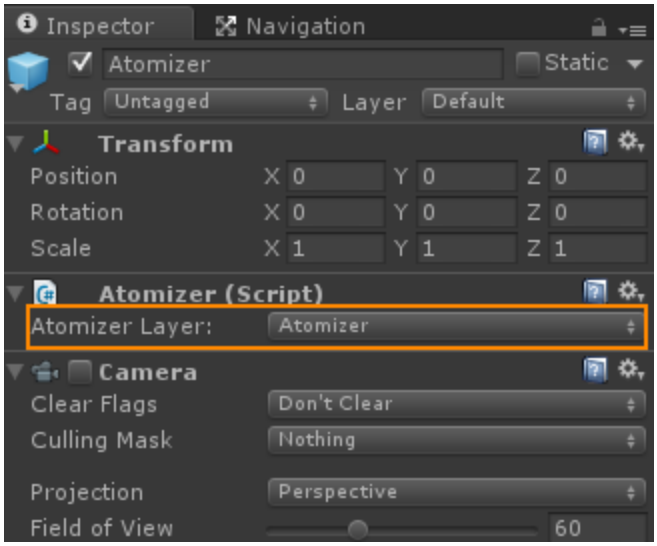
Quick Start Guide

Set the Layer for the Atomizer

After you import the Atomizer package, the Atomizer prefab will be located in the Project tab under Atomizer/Resources.



When you select it and take a look at the Inspector, there is only one setting to worry about, the “Atomizer Layer”.



The Atomizer makes use of a separate layer to render effects, so it's recommended that you create a new layer and assign it to the Atomizer here.

Create an Effect in Code

The first step is to create an "AtomizerEffect" object:

```
AtomizerEffect baseEffect = Atomizer.CreateEffect();
```

Let's then create a "Pop" effect and set some parameters:

```
Pop pop = new Pop();
pop.Duration = 1.5f;
pop.Force = 1.0f;
```

We can then "Combine" the pop into the baseEffect:

```
baseEffect = baseEffect.Combine(pop);
```

Now we can get the Atomizer to atomize a gameObject using that effect:

```
Atomizer.Atomize(gameObject, baseEffect, OnFinish);
```

“OnFinish” is a function that will be called when the effect is completed, so that we can take some appropriate action.

So the basic idea behind scripting the Atomizer is:

1. Call Atomizer.CreateEffect() to get an AtomizerEffect object.
2. Combine (or Chain - covered below) any effects you’d like, setting parameters on each.
3. Call Atomizer.Atomize, passing it the target GameObject and the AtomizerEffect.
4. Get notified that the effect has finished playing in OnFinish and take whatever action you need to.

The included “AtomizeOnClick” script under Atomizer/Example/Scripts demonstrates the range of effects available.

Scripting Guide

Combining and Chaining effects

The AtomizerEffect has two methods for adding an effect: Combine and Chain.
If you call **Combine**, then it will play simultaneously with all other effects in that “combination group”.
For example, let’s say we’ve set up two effect objects: SwapColours and Jiggle (with the objects being named swap and jiggle, respectively). We then Combine them into an AtomizerEffect (named effect):

```
effect = effect.Combine(swap).Combine(jiggle);
```

When we later Atomize that effect, the swap and jiggle effects will play at the same time, and they would be considered part of the same combination group.

If you call **Chain**, it won’t start playing the given effect until all previously queued effects have finished playing. Let’s extend our example above to play a Pop effect (named pop) once the swap and jiggle effects have completed:

```
effect = effect.Combine(swap).Combine(jiggle).Chain(pop);
```

When this is Atomized, if will complete the swap and jiggle animations, then pop. The pop effect is in a separate combination group.

Note that the OnFinish callback you pass when calling Atomizer.Atomize won’t be called until all effects have completed playing.

Creating a new effect script

All of the effects implement the IAtomizerEffect interface.

```
public interface IAtomizerEffect
{
    bool IsFinished
    { get; set; }

    bool IsPlaying
    { get; set; }

    void SetParticles(ParticleSystem.Particle[] particles);
    void Update(ParticleSystem.Particle[] particles, float activeTime);
}
```

SetParticles is called by the Atomizer system when the effect is about to be called into action. It gives you the initial state of the particles. Any setup should be completed in this function.

Update is called every frame until you set **IsFinished** to true. You're given the updated state of the particles, along with the activeTime, which is the cummulative time that this effect has been active.
You can use activeTime to control the effect duration, or some other timing related effect.

IsPlaying is controlled and set by the Atomizer system, so you typically shouldn't need to set it yourself.

Once you've implemented that interface in your new effect class, you're free to pass it to the Chain and Combine methods of the AtomizerEffect object.

Controlling AtomizerEffect

The AtomizerEffect object controls some important options that are used for all applied effects, such as particle size and maximum particle count.

Setting a reasonable maximum particle count is crucial for acceptable mobile performance. This is set through the **MaxParticleCount** property.

Using the Atomizer from JavaScript/UnityScript

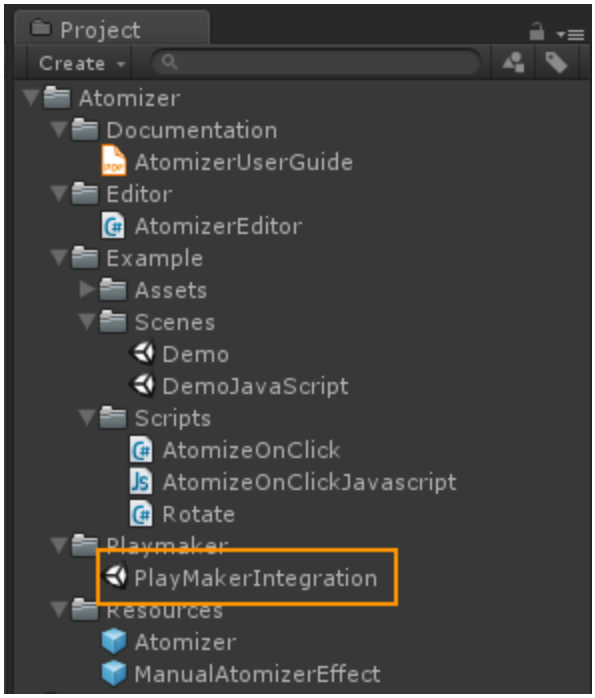
To see how to call Atomizer functions from JavaScript, take a look at the included AtomizeOnClickJavascript script under Atomizer/Example/Scripts.

PlayMaker Integration

Atomizer includes a package that provides basic integration with the PlayMaker addon.

To get started, double click on the "PlayMakerIntegration" package icon under Atomizer/Playmaker (If you imported the Atomizer package then you'll find this in the Unity

Project tab).



Doing so will import several new files under Playmaker Custom Actions/Atomizer.

Each Atomizer effect will be listed under the “Atomizer” category in the PlayMaker action browser. Once you add an effect you can modify any of its parameters without ever having to write a line of code. When the effect has completed playing, it fires a Finished event.

When the Atomizer effect plays, it disables all of the renderers on the target object. If you want to turn them back on in the next state, make use of the AtomizerShow action.

Effect Listing

An effect will typically alter one or more of the following things for each particle:

- Position
- Size
- Colour

With the ability to arbitrarily chain and combine effects, It’s important to understand how the effects will work together. This is especially true in cases where you combine effects that alter the same properties (such as position).
If an effect specifies that it alters a property “additively”, it means that it alters the existing property in a way that can possibly work in combination with other effects. If it specifies that it alters a property “absolutely”, it means that it overwrites the old value entirely.

It’s also important to understand that order matters when combining effects.
For example, if we’ve created our AtomizerEffect and combine three effects:

```
effect = effect.Combine(one).Combine(two).Combine(three);
```

Then although they're combined and running at the same time, in a single frame:

1. one processes the particles, then
2. two processes the particles, then
3. three processes the particles

Which means that effect three has the final say if it alters a property in an absolute way.

BreakAndReform

Description:
Causes the particles to fly apart in random directions, then return to their original position. The first phase where the particles are moving apart is called the Break Phase, the second phase where the particles return to their original positions is called the Reform Phase.

Alters:

- Position (additive in the break phase, absolute in the reform phase).

Parameters:
BreakApartSpeed: Controls the speed of the particles during the Break Phase. Given in world units per second.
BreakPhaseDuration: The duration, in seconds, of the Break Phase.
ReformPhaseDuration: The duration, in seconds, of the Reform Phase.
Note that there's no need for a "reform speed" since it's defined implicitly by the ReformPhaseDuration.

Disintegrate

Description:
Causes the particles to change to a specified colour in a random order.

Alters:

- Color (absolute).

Parameters:
Duration: The time in seconds it takes for all particles to change to the FadeColour.
FadeColour: The colour that all particles will be set to.

Jiggle

Description:
Causes the particles to move a random amount in each direction.

Alters:

- Position (additive).

Parameters:
Duration: The time in seconds the effect will last.
Noise: The limit, in world units, that a particle can move in one frame.

Percolate

Description:
Causes the particles to move continuously in a cardinal direction (Up, Down, Left, or Right). A slight bit of randomness prevents all particles from moving at exactly the same rate.

Alters:
- Position (additive).

Parameters:
PercolationTime: The time in seconds it takes for all particles to start percolating.
PercolationSpeed: The speed, in world units per second, at which particles move once they've started percolating.
Duration: The time in seconds that the effect should play for.
Direction: One of four values to indicate the direction to move the particles: Up, Down, Left, or Right.

Pop

Description:
Causes the particles to fly apart in random directions.

Alters:
- Position (additive).

Parameters:
Force: The force at which the particles will fly apart, in world units per second.
Duration: The time in seconds that the effect should play for.

Pulse

Description:
Causes the particles to pulse (grow then return to normal size), with particles further from the center of the system pulsating a greater amount.

Alters:
- Position (additive).
- Size (additive).

Parameters:
PulseLength: The time in seconds for the particles to reach their maximum size, or return to their normal size from the maximum.
PulseStrength: Scales the maximum size of the particles.
Duration: The time in seconds that the effect should play for. If you want the effect to end with the particles at their original size, set this to some multiple of 2 times the PulseLength.

Smolder

Description:
From a given source position, radiates a new particle colour to all the particles. Then radiates a second colour and causes the particles to move outwards from the source.

Alters:

- Position (additive in the final phase).
- Colour (absolute).

Parameters:
SmolderColour: The colour to set the particles to during the first colour change.
BurntColour: The colour to set the particles to during the second colour change.
EndColour: The final colour to set the particles to.
SmolderDuration: The time in seconds for all particle colours to be set to the SmolderColour..
BurntDuration: The time in seconds for all particle colours to be set to the BurntColour.
Smoothness: Controls how uniformly the particles turn from their original colour to the SmolderColour. Required to be in the range (0.0, 1.0) -> 0.0 < Smoothness < 1.0.
Source: The screen space coordinate to be used as the origin of the effect.

SwapColours

Description:
Swaps the colour channels of the particles in one of several ways.

Alters:

- Colour (absolute).

Parameters:
ColourSwapMode: The method to use to swap colours. This can be to swap any or all of the red, green, and blue colour values, or to invert the colours.
Duration: The time in seconds to complete the swap.

Vacuum

Description:
Moves all the particles towards a given world coordinate, with particles near the center of the

particle system moving quicker than particles towards the edges.

Alters:

- Position (additive).

Parameters:
VacuumPoint: The point, in world coordinates, that the particles will move towards.
Duration: The time in seconds that the effect will play.
MoveSpeed: The speed in world units per second that the particles will move towards the VacuumPoint. Note however that the movement amount is also scaled by the distance from the particle to the VacuumPoint (particles closer to the center of the particle system move more quickly towards the vaccum point).

Warp

Description:
Instantly moves most of the particles in a system to another position, with the rest following after.

Alters:

- Position (absolute).
- Size (absolute, although at the end of the effect the particles have returned to their original size).

Parameters:
WarpTime: The time in seconds for all of the particles to have transitioned to their new warped position.
TargetPoint: The world coordinate to warp to.
StartScale: The scale of the particles at the start of the effect. At the end of the effect, the particles have returned to their original size.