User Guide

Setup

Requirements

Unity Game Engine - The package has been tested most recently in 2017.1 but has also previously been run in 2018. Downloads are available on the Unity website

Weather System Install

First up - make a copy of your current project if this isn't going in a blank project, it can't hurt and means that if something somehow goes wrong you can just start over.

1. Download the most recent release of the weather system
2. Either create a new unity project or open yours. In your unity project, click `Assets > Import Package > Custom Package...` then open the release you just downloaded
3. That's it, you're done!

Using the Package

The `Examples` folder is a good place to start. Especially for understanding the structure of the `ScriptableObject`s used by the weather system.

Documentation

Documentation can be found here, enjoy!

Customisation

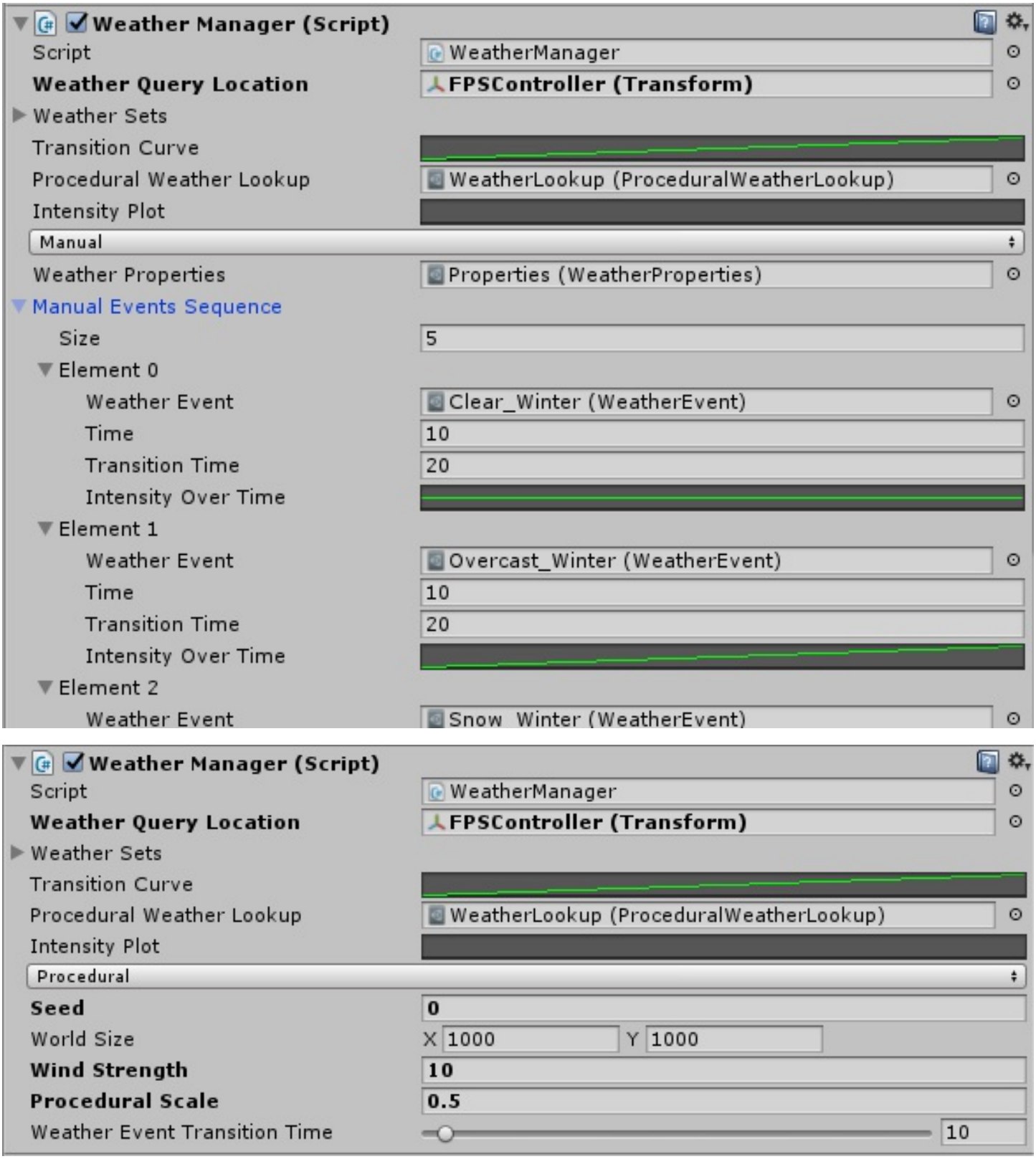
The system provides all the logic for procedural and manual weather sequences. All you have to provide are the visual elements, whatever they may be, to be controlled by some derived `IntensityDrivenBehaviour`. Several example custom control components are provided for shaders and particle effects. The shader components are unlikely to work with an alternative shader (although the example component should go a good ways to providing how you could go about creating your own), however the particle effect controller is likely to be reusable in alternate circumstances.

Weather events, properties and the lookup table for temperature/humidity->weather type are all `ScriptableObjects`. Therefore, to create a new weather event, just add the new weather type from the “WeatherSystem->Weather Types” menu item and then create a new `WeatherEvent` by right-clicking in a project folder and clicking “Create->WeatherSystem->WeatherEvent”. Other weather system items are also available under this menu - Weather sets, Weather lookup tables, and various weather properties.

Modes

The system can operate in two modes. **Procedural** and **Manual**. This is

selected from the `WeatherManager` component.



In manual mode, a sequence of pre selected weather events is cycled through. This mode is good for scenes in which the weather needs to be manually curated. If, on the other hand you want to have dynamic,

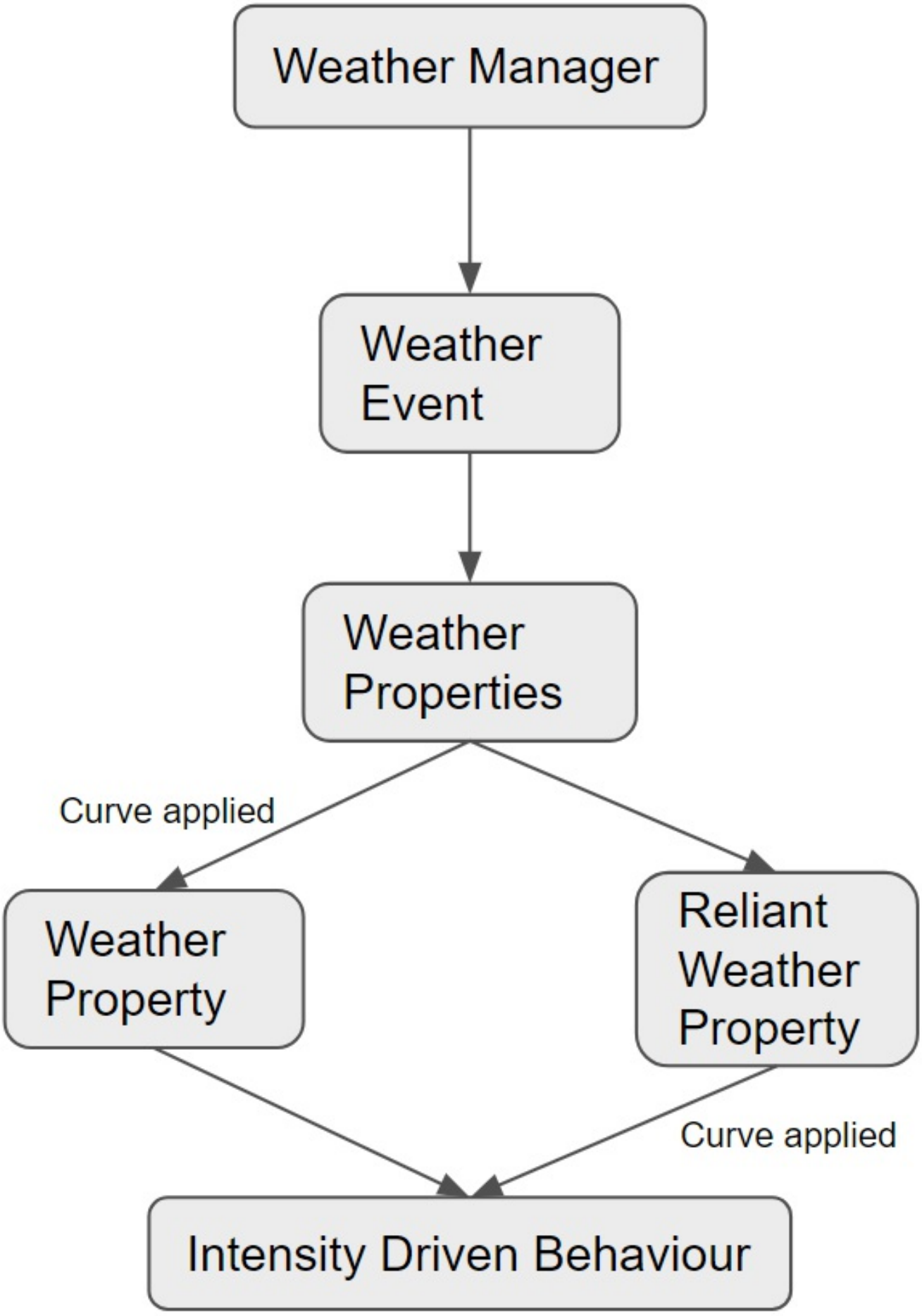
generated weather Procedural mode is for you. Leave the `Seed` calue at zero on `WeatherManager` to have a random see or enter a value for a repeatable generation. Seeds are debugged to console on each generation, in case you come across a particuarly interesting generation - this is also useful for testing.

Structure

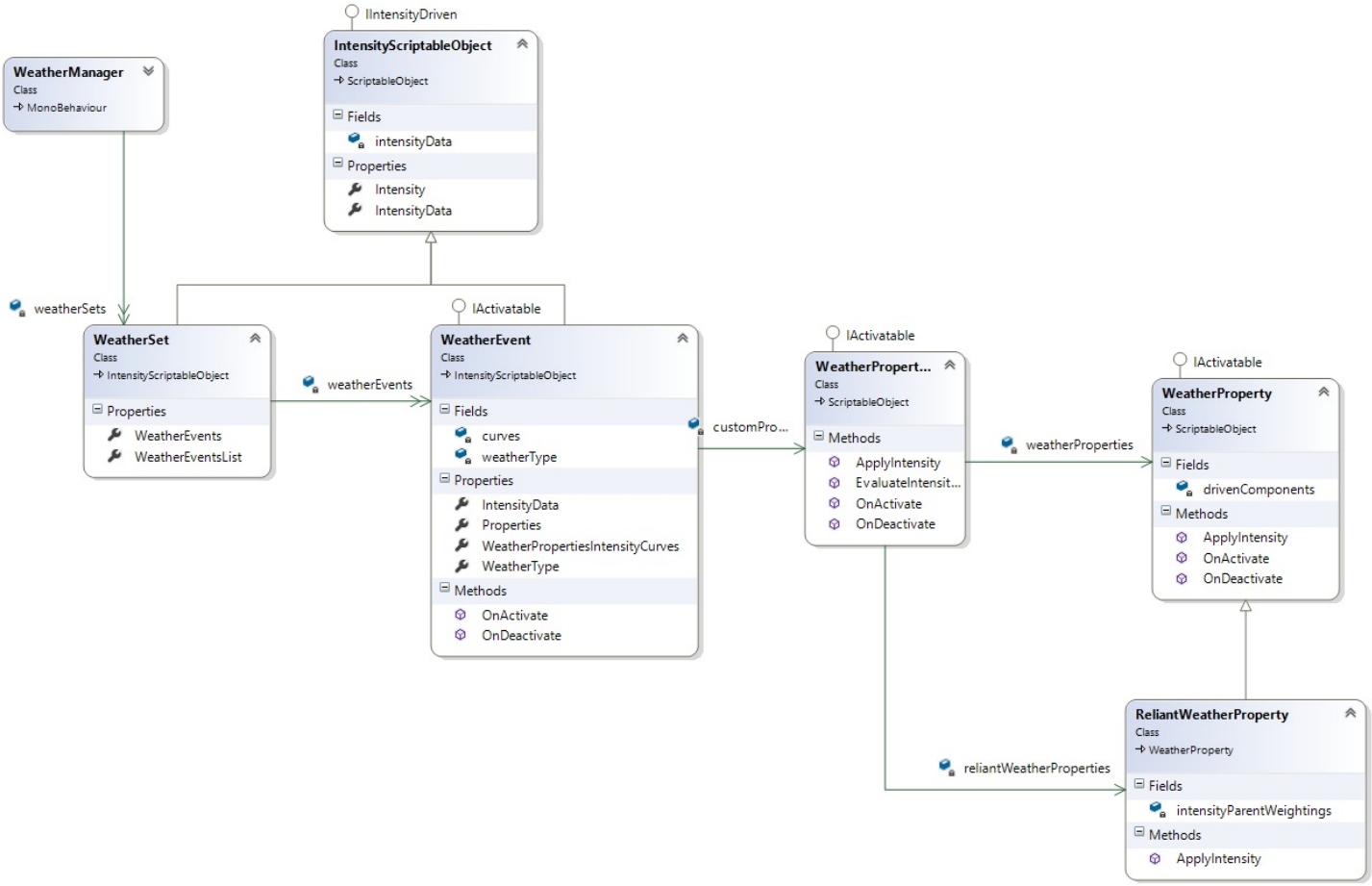
In procedural mode, the weather is determined by a lookup table `ProceduralWeatherLookup`. A humidity and temperature value are generated and then converted to an enum. These enums are then used to lookup the corresponding `WeatherType` that should be active. If this weather type does not match the currently active `WeatherEvent` then a transition is started into the new `WeatherEvent` contained in the `WeatherManager`'s `WeatherSet`. Lookup tables are scriptable object and so can be easily swapped out for an easy change to the weather. For example, a summer and winter lookup table could be used for creating a simple season system.

The system is built around a tree-like structure in which an IntensityData object is propagated from the `WeatherManager` to weather-property-specific (eg. precipitation) components derived frin `IntensityDrivenBehaviour`. The scriptable object structure is very similar/identical to the flow of data to `IntensityDrivenBehaviour`s in the scene, which implement the control of the visuals for the weather. These are connected to their parent `WeatherProperty` at runtime.

Intensity Data Flow



Or, if you're a UML diagram kind of person then:



As you can see, the `WeatherManager` `MonoBehaviour` needs a `WeatherSet`.

`WeatherSet`s are just a collection of `WeatherEvent` objects which in turn hold a `WeatherProperties` object; a collection of `WeatherProperty`s. And then finally there's `ReliantWeatherProperty`s which inherits from `WeatherProperty` in order to allow for weather properties which rely on a number of `WeatherProperty`s to calculate intensity values, for example a Visibility may rely on a

IntensityData

Documentation

`IntensityData` is an object that is passed down the hierarchy and is modified at various levels by `AnimationCurve`s. It includes the intensity of the current weather (0->1), the current Temperature (as an enum) and the current Humidity (as an enum) among other information. Further data can be queried from the `WeatherManager`.

Intensity Driven Behaviours

Documentation

`IntensityDrivenBehaviour`s are the method by which to control your weather effects - simply create a new script and rather than inheriting from `MonoBehaviour`, inherit from `IntensityDrivenBehaviour`. All you then need to do is override `UpdateWithIntensity`, `FadeOut`, `OnActivate` and `OnDeactivate` to implement your desired behaviour. `IntensityDrivenBehaviour`s can be driven by a single `WeatherProperty`. If you find yourself needing two or more `WeatherProperty`s to drive your behaviour you probably need a new `ReliantWeatherProperty` which relies on those `WeatherProperty`s.

Conditional Behaviour

Documentation

If your behaviour should only run when a certain condition is met then rather than inheriting from `IntensityDrivenBehaviour` inherit from `ConditionalIntensityDrivenComponent` and override the `ShouldUpdate` method. `ShouldUpdate` should return true when... you guessed it, the behaviour should update. In order for your update code to only be run when the condition is met, update code should go in `ConditionalUpdateWithIntensity` rather than `UpdateWithIntensity`.

All other functionality should be as with `IntensityDrivenBehaviour`.

If the condition is that a the active `WeatherEvent` is of a certain `WeatherType` then the work's been done for you! Check out the `WeatherTypeSpecificIntensityDrivenBehaviour` in the documentation.

Instance Events

Documentation

If you have a behaviour that you wish to have a (deterministic) chance to happen occassionally (think thunder) then you want to include a `IntensityDrivenInstanceEvent` in your scene and implement a script that derives from `InstanceEvent` or implements the `IInstanceEvent` behaviour. Once your implementation is complete, add your script as the "Instance Event" property on the `IntensityDrivenInstanceEvent`.

`IntensityDrivenInstanceEvent` derives from `WeatherTypeSpecificIntensityDrivenBehaviour` meaning you can limit your instance events to certain weather types.