Blendables

Blendables assets can be smoothly interpolated and used to affecting the rendering (post processing, fog, Ambient Cubemap, ambient occlusion).

A **Blendable** is an asset that has properties which can be smoothly interpolated (blended) with other blendables. We mostly used blendables for PostProcessMaterials but the system can be used for anything that should be depending on the view (usually dependent on the camera position).

Blendable

We have **Blendables** in the engine for a while but only used for PostprocessMaterils / PostprocessMaterialInstances. But the concept is more general purpose as it allows to blend arbitrary data (linear values or colors are best suitable) to some final data. Any subsystem can pickup the data in the view and affect rendering. As the data is blended per view it means in the splitscreen case you can have different settings blended for each view (e.g. hit indicator affecting post process).

A **Blendable** is an object that has the IBlendableInterface and currently that is implemented by those asset types:

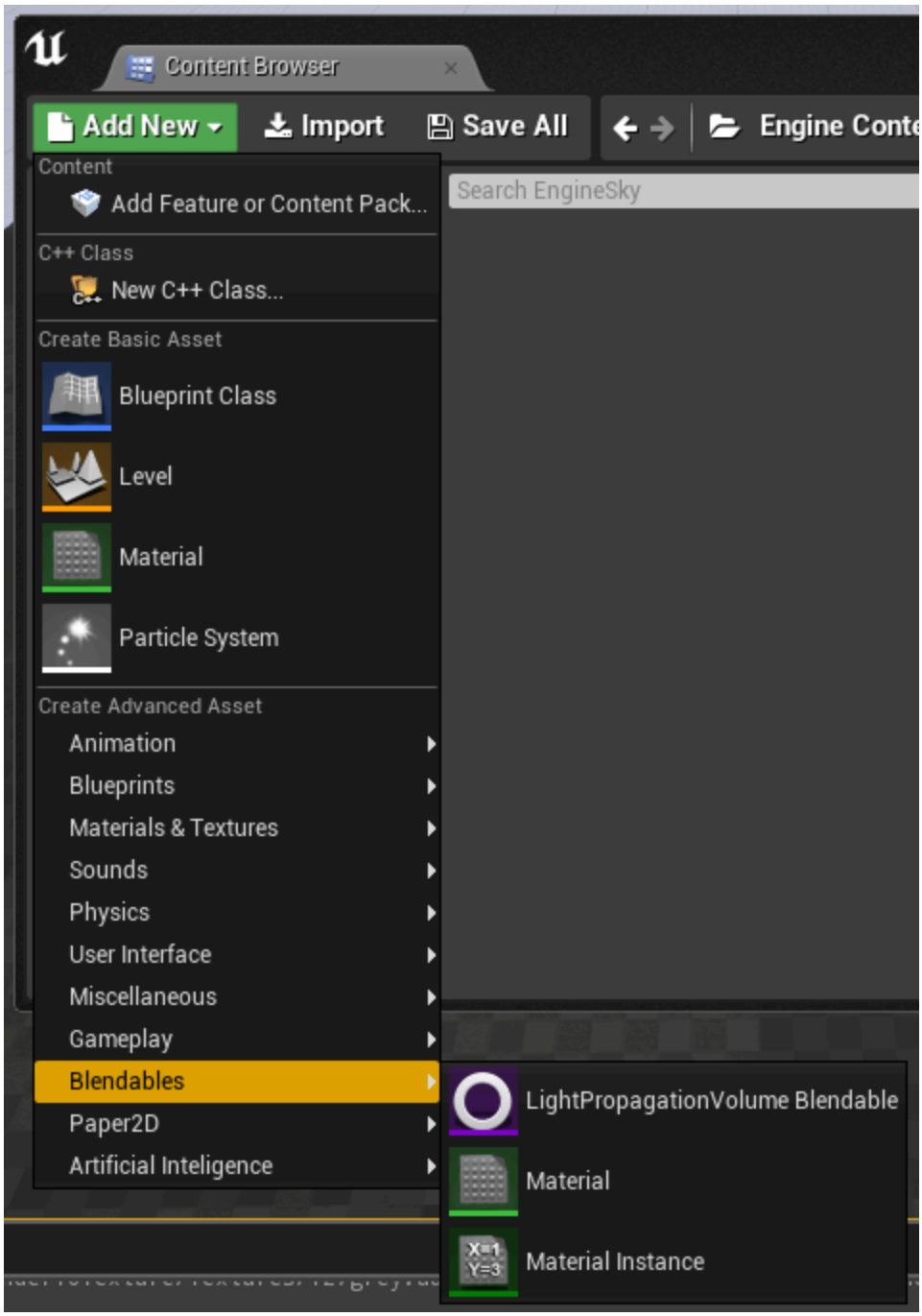
- PostprocessMaterials
- PostprocessMaterialInstances

- LightPropagationVolumeBlendable (see below)

The **Blendables** container can be found in the PostProcessSettings which is embedded in the following objects:

- PostProcessVolume
- PostProcessComponent
- SceneCaptureActor
- CameraComponent

The **LightPropagationVolumeBlendable** asset was created to demonstrate on how to create new blendables and to show how we can replace the existing PostProcessSettings. The existing system worked well when it was small but the large amount of settings showed the need for a more sophisticated system.



Blendable assets show up in the content browser in the Blendable category. You can use the Add New or filter asset by this category.

The benefits of the new system:

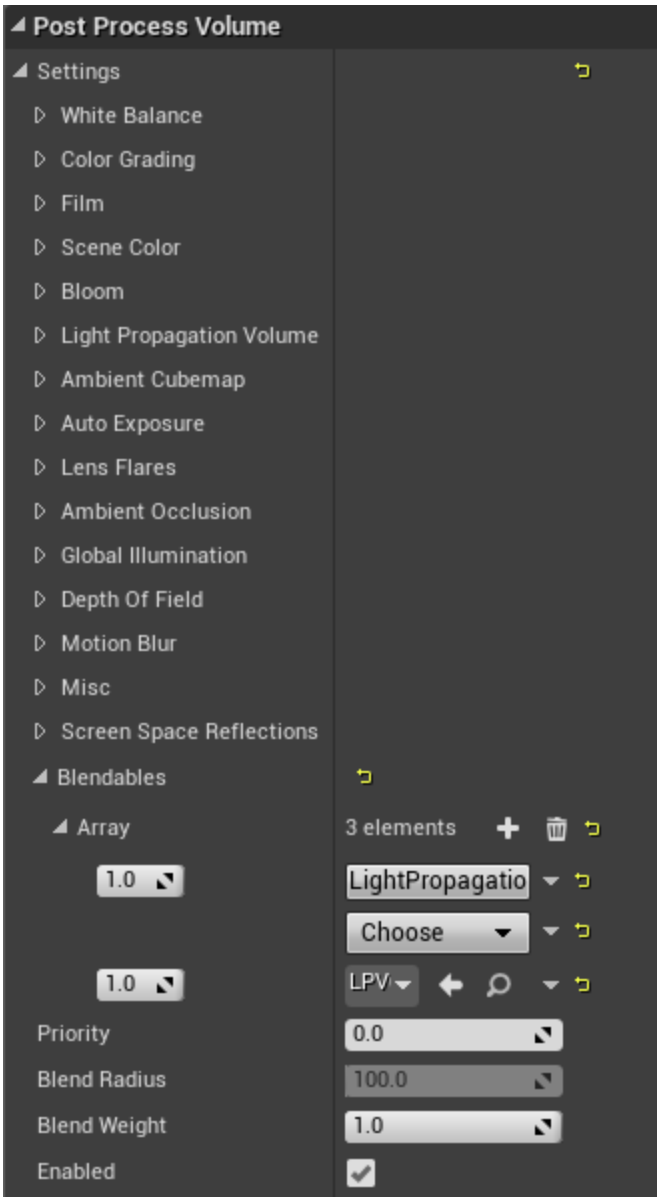
- Easier to extend and maintain with engine changes (no need to change the one central structure, can be in it's own module)
- Indirection over packages allows to adjust the content without level access (version control)
- Indirection means a single asset can be reused in many occasions (less redundancy, more power)
- Custom UI for each blendable is possible (much harder to do with a single struct)

- Each Blendable reference can have its own weight, the asset can have a weight (see LightPropagationVolumeBlendable) and a per-property-weight would be easy to do.
- Breaking up the large struct make interaction with Blueprints more efficient and simpler.

The Blendables Container

The container is implemented as an array of weights and Blendable Interface references.

When you open the PostProcessVolume setting and look at the blendables array you see an array of weightes with references to a blendable asset. The weights are usually in the 0..1 range and the reference can be to an asset living in a package (created with the content browser) or living in the object that contains the blendables array.



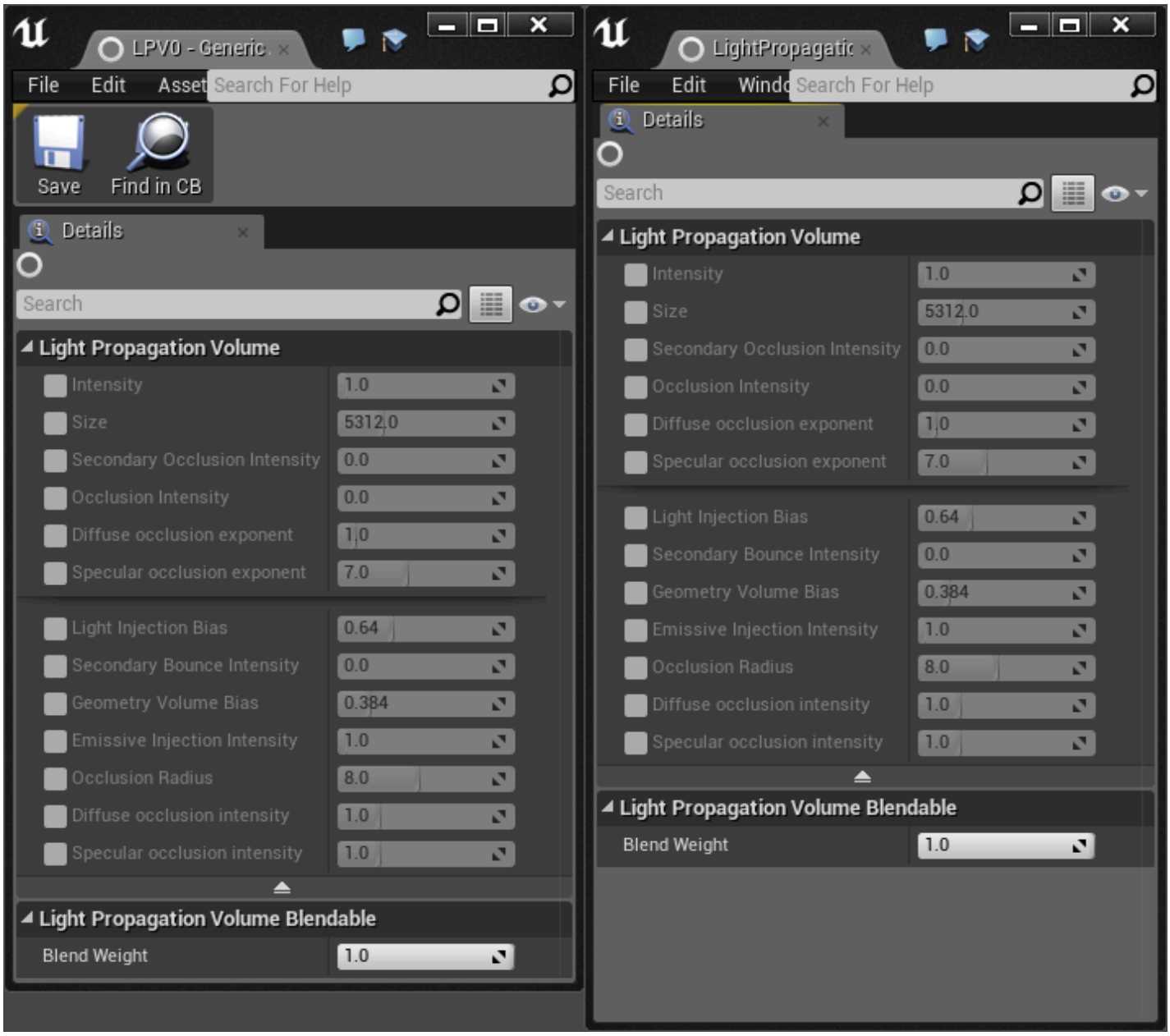
The blendables container can be found in the post process settings (here in a postprocess volume). The array here has three elements, a LightPropagationVolumeBlendable (living in the volume object), a not yet used array element and a reference to a an asset (living an in a package) called LPV0. The weights are 1.0 for both blendables.

When you create new elements in the array you can choose to create a blendable of a specitic type or using an asset reference (e.g. Material, Material Instance). Over time we intend to create more blendable types (e.g. Bloom, SceneColor, DepthOfField, ...). The reference can be of any type that is a Blendable (implements the IBlendableInterface). The order in the array is opposite to how layers stack because their blending is applied from top to bottom and blending is overwriting the data that was there before. Keep in mind the data of many volumes (or other objects) are getting combined taking weight and priority into account.

Note: It's a good practice to have a unbound PostProcessVolume with lower priority in the level called "global". To get get full control over an existing level you can add an unbound volume with high priority. To check if a blendable has effect you can quickly adjust it's weight to 0 and back.

Blendable in the package, as part of the object (e.g. volume) or dynamically created in the Blueprint

It's your choice but we suggest the package (referencing a named asset in the package) as it allows for easier bulk adjustments later on minimizing version control conflicts. For maximum programability it's possible to create a blendable in a Blueprint. As Blueprints are a form of programming this is simialar to putting settings in a UI or hard coding them in the code. The code method is more involved and makes it harder for others tweaking the settings.



The LightPropagationVolumeBlendable details as they can be seen in the editor. No matter if the blendable was created in the content browser (left) or created in the object (e.g. post process volume) the user interface is similar. It's good practice to give each property a checkbox (weight=0 or weight=1) and give the whole struct a blend weight.

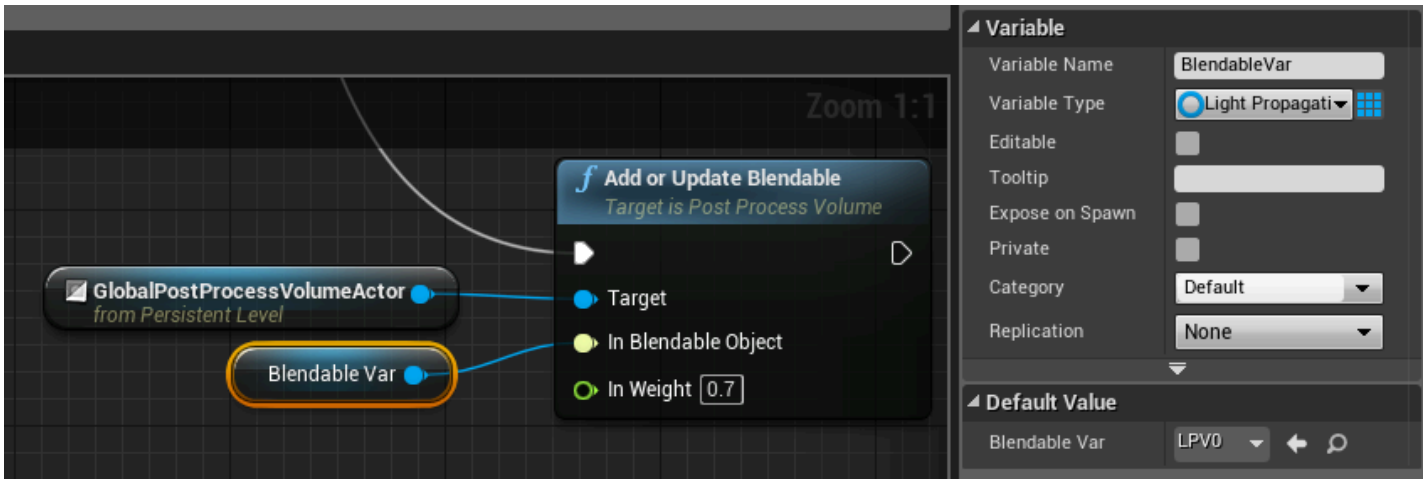
How to create your own Blendable (in C++)

At the moment we suggets to copy the LightPropagationVolumeBlendable plugin. After creating the asset you can pick up the blended data the same way the Light Propagation System is doing that. The method **GetSingleFinalDataConst()** is used to get the data after it was blended. For best performance you might want to avoid calling this function unneccessarily (too often).

Blueprint

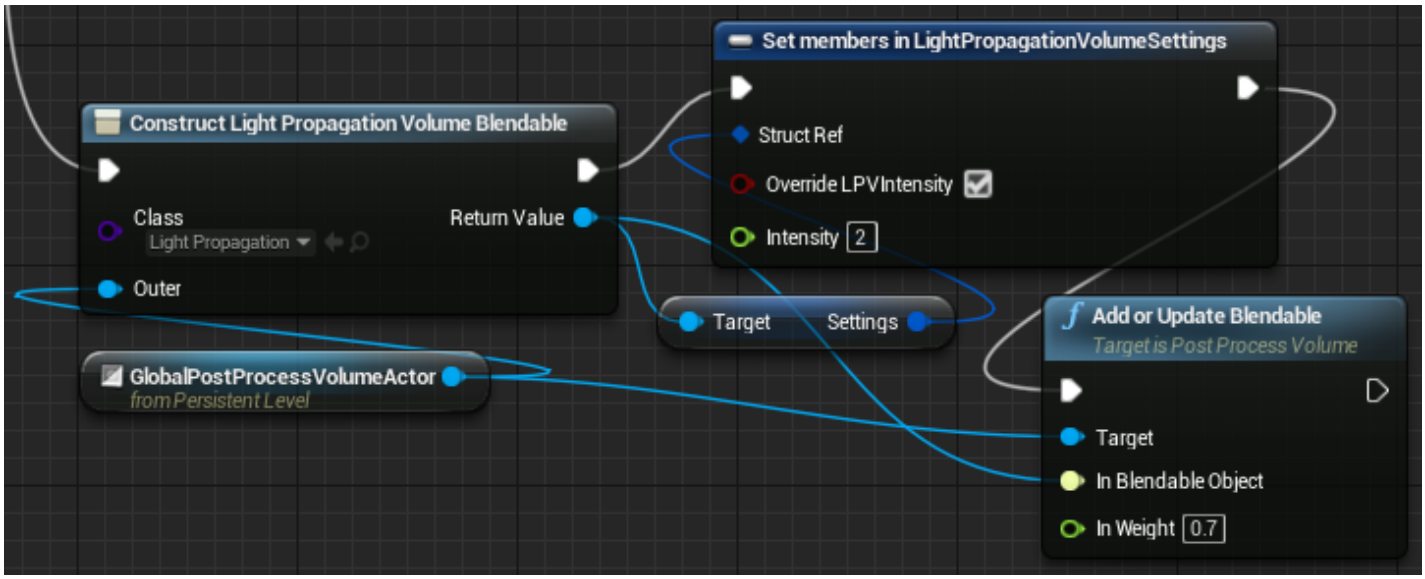
The **AddOrUpdateBlendable** Blueprint function is exposed where you can find PostProcessSettings. It allows convenient access to the blendables container. You pass in the object that has the Blendables container, the weight and a reference to the blendable. If the reference was already found in the container it simply updates the weight. It doesn't remove container elements as that could confuse other code traversing the container and has implications to the garbage collection. There is no real performance cost a blendable reference of a weight of 0 as removing the element should never be needed.

Here you can see how to reference a blendable asset in the content browser:



The variable 'BlendableVar' of the type LightPropagationVolumeBlendable (Object Reference) is used to reference an LightPropagationVolumeBlendable asset called 'LPV0'.

With the **ConstructObjectFromClass** Blueprint function you can create a new blendable in the Blueprint. By setting the **Outer** of the new object to the object that has the blendables container you get the same behavior as if you would created the object in the UI (create the blendable as part of the object).



Here we create an object of the type LightpropagationVolumeBlendable, get the settings and set some members with the SetMembersIn... function.

> At the moment you need to manually put the override flag to true (checked checkbox), otherwise the property with the same name would not get picked up.

Future

- The context sensitive feature when browsing the function AddOrUpdateBlendable doesn't work yet (workaround: disable the 'context sensitive' checkbox).
- We intend to break up all PostprocessSettings into object like the LightPropagationVolumeBlendable so one day the PostProcessSettings can be removed. Old levels can be converted on load without any data loss. In order to avoid a lot of assets spaming the content packages we would create the objects as part of the level.
- We want to further polish the Blueprint interaction to make it easier to use.
- We could easily expose a Blendable array in the world settings and the project settings.
- In order to get more transparency which Blendable are getting applied we should have some debug view showing the weights and asset/object names and types.