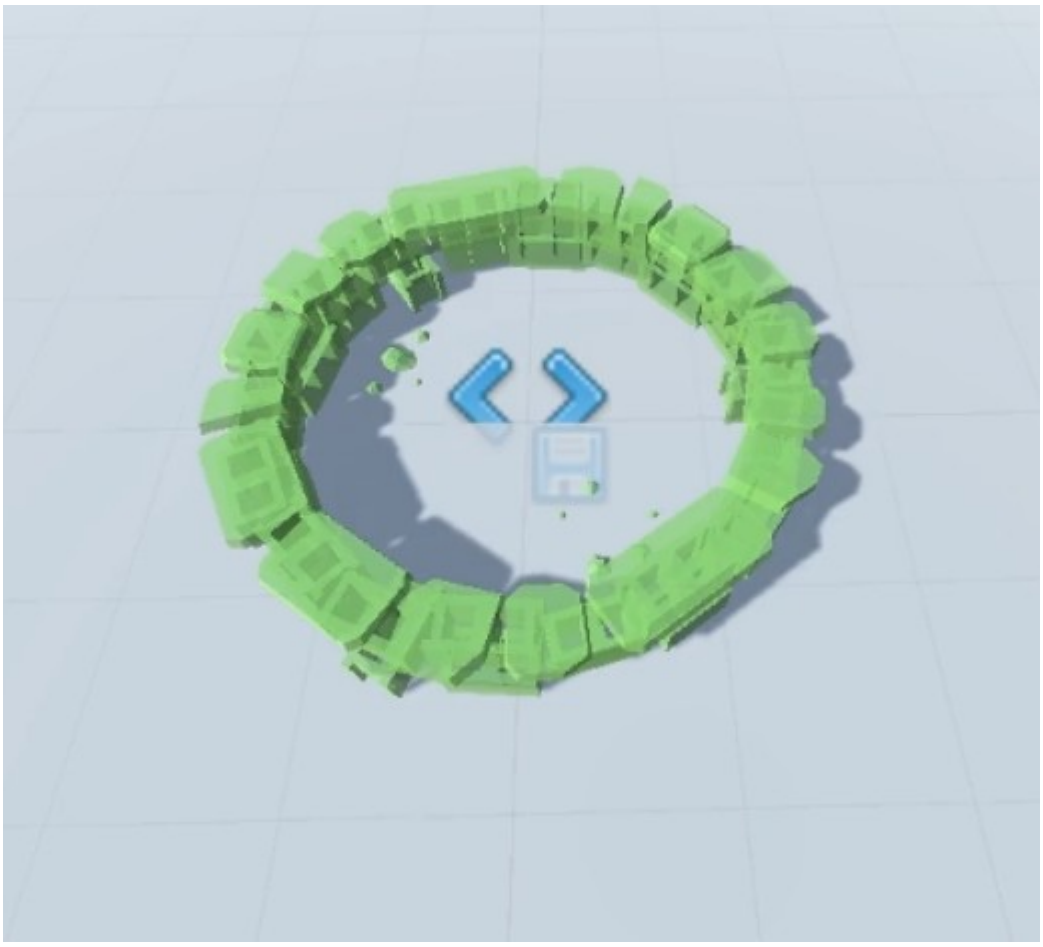
easyPreview



by protowave

CONTENTS

1. INTRODUCTION

Thank you for downloading easyPreview, This asset is a single material solution to provide player with feedback when manipulating a prefab, it could be as simple as turning the model into a ghost, or changing the color based on placeable or uneven terrain. When an item is previewed, any active colliders are disabled as to not interfere with raycasts and collide with any rigidbodies in the scene, on release, the disabled colliders are re-enabled.

If you find this asset useful, it would be appreciated if you could leave a review, Thanks.

2. PIPELINE SETUP

The project you downloaded from the asset store is configured for the Built-In pipeline, if you are using **URP** / **HDRP**, you need to upgrade the materials for said pipeline, this can be done by navigating to;

easyPreview/Materials/

then select all the materials in this folder;

once selected, select the Edit menu > **URP** / **HDRP** > Upgrade Selected Materials to **URP** / **HDRP**

at this point, the provided material should be good to go for your pipeline.

3. PREVIEW COLORS & DATA

Now the material is set up for your pipeline, you'll want to know how to set its colors, head over to;

easyPreview/Resources/

and select the "**previewData**" scriptable object you find there.

Click the dropdown titled "**Preview Colors**" and you'll find color pickers for the 4 provided states, these colors support Alpha transparency, however, if your material is set up to be Opaque and not Transparent, this will be ignored.

In the previewData scriptableObject you'll also find;

StartColor – use this dropdown to select which color should be the default starting color of the preview material. You'll likely change this based on the situation in game, however, if you decide to only use a single color for the preview, this would be how you'd set it and forget it.

Color Change Duration – this float represents the time in seconds the transition between colors will take, setting this to 0 will make the change happen instantly, when no objects in the scene are using the preview material, the color change will ignore whatever is set here and happen instantly.

Preview Material – this is the material easyPreview should use, if you want to use a custom material and not the one provided, this is where you'd set it.

4. SUPPORT FOR CUSTOM MATERIALS & SHADERS

easyPreview supports any material that makes use of the default material**.color** accessor, this includes a variety of the material shaders provided in each render pipeline. Support for your custom materials / shaders can be configured by setting its color property name to "**_Color**" as outlined in the Unity Documentation ***here***. An example URP shader named "**ex_URP_CUSTOM**" is provided in the "**Shaders**" folder, it has the properties;

Color; changes the base material color.
FresnelSize; changes the size of the Fresnel effect.
FresnelTint; this controls the blend between the "Color" and the "FresnelTintColor'.
FresnelTintColor; the color of the Fresnel effect, this is blended with the "Color" parameter.

5. SCRIPT DESCRIPTIONS

"preview";

The static class at the heart of easyPreview, it's what you'll send data to to start and release GameObjects from their preview state.

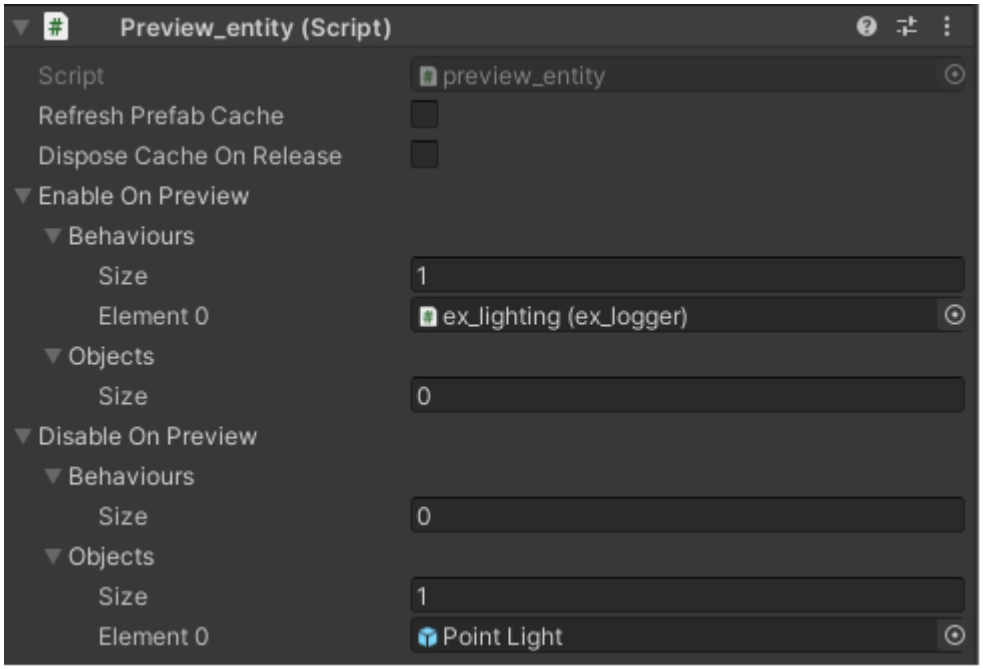
"preview data";

The class the "**previewData**" scriptable object that holds color and other data derives from.

"preview_mat_manager";

This class will get created and placed into your scene automatically with the name "**easyPreview_pmm**", it's used to manage the material in play mode, as we need a Monobehaviour to animate the material color over time.

“preview_entity”;



This is the class that'll be placed on all objects when they are previewed, this happens automatically when you call for a preview, but can also be done manually in the inspector if you'd like to make use of some of its extra features listed below;

“Refresh Prefab Cache**”**;

When calling to preview an item, a cache will be made of all the active Colliders and then they will be disabled. You may have an item that is placed and picked up multiple times, and between these events, different parts of the prefab could become enabled / disabled, if so, setting this bool to true will force the cached colliders and Renderers to update to the new prefab state, while false will just use the originally calculated cache.

“Dispose Cache On Release”**;**

This bool controls whether this “**preview_entity**” sticks around after placement, false will keep the script around ready for the next time this item is called to be previewed, true will Destroy() this “**preview_entity**”. False would be useful if you have items that can be picked up and placed multiple times, where true might be useful if you know you won't need the information again, keep in mind, if you delete the cache it can still be auto generated if you do call for a preview again, but any custom data as outlined below will be lost.

“Enable On Preview”;

This is a struct that allows you to add Monobehaviours and GameObjects that you'd like to be enabled when an item begins previewing, it could be a behavior that sends a debug OnEnable(), when the preview is released, anything set here will be disabled on preview release, (Objects / Behaviours have to be disabled before calling to return to that state)

“Disable On Preview”;

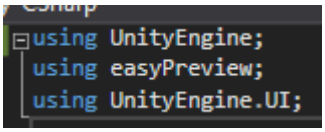
The same as above but in the opposite direction, useful for parts of your prefab that wouldn't make sense to be active in preview mode, an example named “**ex_lighting**” is included in the prefabs folder, where the prefab contains a light that is disabled in preview mode, like the above, only items that were enabled before the preview were called will be disabled and then re-enabled on release of preview.

“ex_placement / ex_logged / ex_mouseOver”;

Example scripts, these will be covered under the “Example” heading further down.

6. NAMESPACE & FUNCTIONS

To use easyPreview, either add "**using easyPreview**" to your class;



or start your function calls with "**easyPreview.**", examples for both below;

without using;
easyPreview.preview.functionName();

using;
preview.functionName();

Now you're set up and know how to call easyPreview functions, we'll cover the functions you can call, and the parameters that can be passed in.

startPreview()

```
startPreview(GameObject _object, previewMode _mode = previewMode.single, bool _refreshPrefabCache = false)
```

This is how you'll tell an object to become a preview,

_object is the object you'd like to begin previewing, this has to be passed in.

_mode determines if previewing an object will wipe the previous preview, if it still exists, this doesn't have to be passed in as it defaults to "**previewMode.single**", the other option is "**previewMode.multi**" which will allow you to have multiple items being previewed at one time as shown in the example scene.

_refreshPrefabCache does the same as the before (**preview_entity**) mentioned Dispose Cache On Release, this doesn't have to be passed in as it has a default of false, call true if you'd like to re-determine the active colliders at the time of the function call.

releasePreview()

```
releasePreview(GameObject _object = null, float _delay = 0f, bool _disposeCache = false, GameObject _ignoreItem = null)
```

To release a preview, use this function, it has a lot of variables you can pass in, however, none are necessary but will provide different results based on the current previewMode, there are also multiple simplified versions of releasePreview that can be called, but first, an explanation of each variable..

_object is the specific object you'd like to release, this has a default value of null and if left as null, will release ALL objects that are currently using the preview material, this means in **previewMode.single**, the currently previewed object will be released, and in **previewMode.multi**, ALL previewed objects will be released.

_delay is a float the represents time in seconds you'd like to delay the release of the preview material, this could be used in conjunction with a particle system, as the change could be delayed for an amount of time to allow the particles to fully obscure the released item from the player, the default value of this is 0 and doesn't need to be passed in, 0 causes the change to happen instantly.

_disposeCache is the same as the "**Dispose Cache On Release**" of the "**preview_entity**" class, the default value is false, however, if the entity was set up in the inspector to dispose, that value will take precedent.

_ignoreItem allows an item to be ignored from the release process, an example being building a fence in game, you have 6 fence posts already placed, and a 7th currently held that also has the preview material, you could confirm the current fence and release their materials while the held item retains its preview material, the default value is null meaning no item is ignored..

As described above, the releasePreview function has some some overrides and variations that can be called, these are;

```
releasePreview(GameObject _object)
```

object only, the other items will retain their default values

```
releasePreview(float _delay)
```

delay only, the other items will retain their default values

```
releasePreview(bool _disposeCache)
```

dispose cache only, the other items will retain their default values

```
releasePreviewIgnoringObject(GameObject _ignore, float _delay = 0f, bool _disposeCache = false)
```

release all previews ignoring the passed in GameObject, this gets a slightly more verbose name to differentiate it from the above object only call.
previewColor()

```
previewColor(previewColor _type, Color32? _customColor = null)
```

This is how you'll update the color of the preview material, you'd call this, for example, from a script that checks the hit.normal of a raycast hit to determine if the ground is flat enough to place an object, the types you can pass through are:

previewColor.canPlace

previewColor.cantPlace

previewColor.ghost

previewColor.custom

The function also has an optional variable named _customColor that default to null, this variable does not have to be passed in, however, if you do pass a color in here while also passing in the _type **previewColor.custom**, then the custom color set in the inspector "**previewData"** will be ignored, and the _customColor will be used instead, this allows you to use more than the 4 preset colors, if you so wish.

With the 3 above mentioned functions, you'll have access to all the features provided by easyPreview, an example scene is provided, which we'll quickly go over below.

7. EXAMPLE SCENE

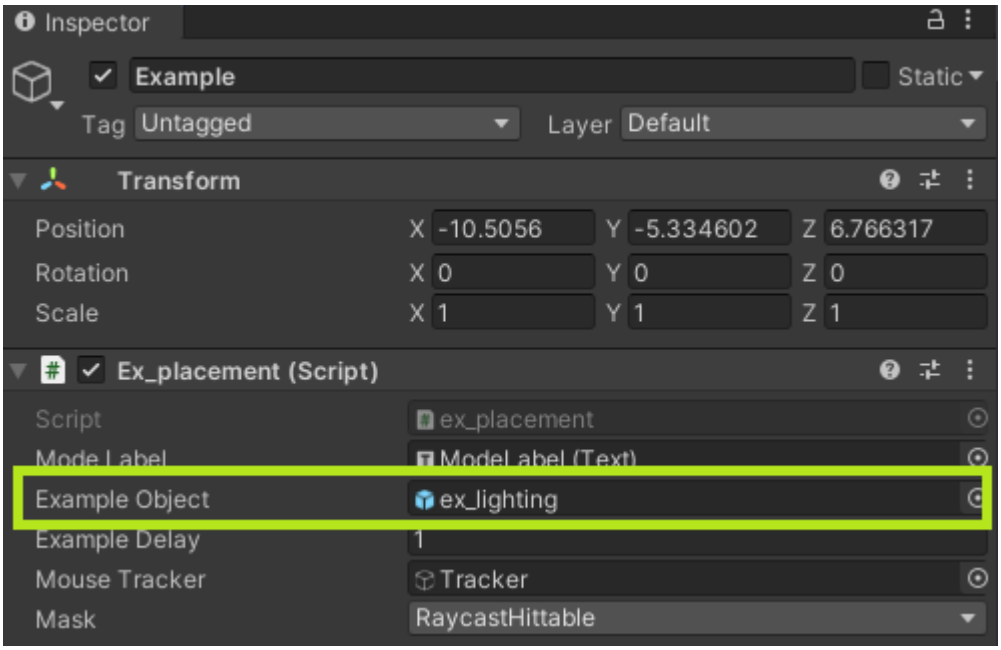
This package contains an example scene named "**ex_previewScene**" that uses the Old Input System, it uses the following scripts;

"**ex_logger**" – debugs to the console when it gets enabled and disabled, and is used on a prefab named "**ex_lighting**" to demonstrate the enable/disable on preview functionality.

"**ex_mouseOver**" – detects when the mouse hovers over a specific pad, on hover, will report to the placer script and change the material.

“**ex_placement**” – handles the spawning / previewStart and releasePreview of prefabs, allows changing between single and multi placement modes.

The scene UI contains the binds required to change between modes and place / discard / confirm item placement.
The scene contains an GameObject named example, on this object you'll find the “**ex_placement**” script, by default it is set to use the “**ex_lighting**” prefab which demonstrates the disabling / enabling of a monobehaviour and a light, if you'd like to swap this prefab out, either for the provided “**ex_multimesh**” prefab, or a prefab of your own, then drag the new prefab into the “**Example Object**” inspector field as highlighted below.



All items contained within this package that have the prefix “**ex_**” are not required for the easyPreview to function and are safe to be deleted after the example is no longer needed.

8. LINKS

- Email: support@protowave.xyz

- Youtube: https://www.youtube.com/watch?v=YMygI1rRVWw&list=PLklWm0q8FMPM2j1em0VBGkyJq4JXgQIJE&index=1