Asset Variants

You can contact me at precisioncats@outlook.com

Table of contents:

Setup:

(You might need to Reimport Precision Cats/)

If you have a previous version of ExtendedScriptableObjectDrawer.cs, you’ll want to delete it.

The Setup tab in Window/Asset Variants gives quick access to most of the basic setup options:

- Setup Minimal: No color wraps, sets the ERSettings asset’s neverUseDefaultMargins to false.
- Setup Grayscale: Activates the grayscale color wraps preset (Color Wraps 3 - Grayscale). Sets the ERSettings asset’s neverUseDefaultMargins to true.
- Setup Random Colorful: Chooses by random one of the 5 default non-grayscale color wraps presets. Sets the ERSettings asset’s neverUseDefaultMargins to true.

To choose another color wraps preset you can search: "Color Wraps" and press -> Toggle Active Sibling <- on the preset you want to try out (or you can use Window/Asset Variants/System Prefs/Color Wraps). If you want to create your own preset, you can use the Duplicate button on the bottom.

- Set for ___: sets the combination of scriptableObjectOnly and materialOnly bools in the AVSettings asset, whitelisting the asset types that can be asset variants. Both will become false if you press Set for All (Default)

- [Enable/Disable] BasicSubAssetsParentEditorReplacer: Activates/deactivates BasicSubAssetsParentEditorReplacer.Priority

- [Enable/Disable] Editor Replacement: If Editor Replacement is disabled, then almost everything Asset Variants does will be disabled (except menus and e.g. calls to AVUtility).

- [Add/Remove] EDITOR_REPLACEMENT: Removes or adds the scripting define symbol EDITOR_REPLACEMENT to effortlessly disable/enable the Precision Cats/editor-replacement/ package entirely. Works almost the exact same as [Enable/Disable] Editor Replacement, or deleting the folder (non-permanently).

Cleanup:

After possibly testing /Asset Variants Examples/, I'd recommend keeping the /Properties/ subfolder, and deleting the other subfolders.

For information on the files in that folder, check out:

/Asset Variants Examples/README.txt

In /Inspector Examples/ I'd recommend deleting the /Testing/ subfolder.

For information on the replacement examples read the Notes fields of their /*.Default assets, and/or check out:

/Inspector Examples/README.txt

Settings:

BecauseAsset Variants is modular, there are 3 settings asset files: ERSettings for Editor Replacement, PDRSettings for Property Drawer Replacement, and AVSettings for Asset Variants.

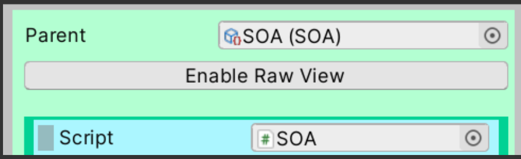
You can find most of Asset Variants' customization available in the Window/Asset Variants window, or in the project settings: Edit/Project Settings.../Asset Variants.

Hovering on the individual settings should give some helpful tooltips.

The window Window/System Prefs, found also at Window/Asset Variants/System Prefs, is useful for controlling the active/inactive and priority (sequence) states of Editor Replacement's and Property Drawer Replacement's Replacers.

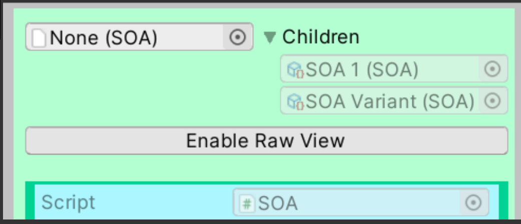
> The red elements are "invalid", this can be if Unity is in dark mode and a ColorWrapEditorSettings asset is for light mode only. Darker colored elements are "Disabled"/"Off", which means the SystemPrefsState asset will not set their systemActive and systemPriority values to EditorProjectPrefs.
>
> Modifying the values in the "States" tab will modify the SystemPrefsState assets themselves, but modifying the top (foldout) lines (the "keys") in the State Keys tab will modify EditorProjectPrefs' values. This is usually preferable because an update to the Asset Variants package can't overwrite your PlayerPrefs/EditorPrefs.
>
> Note that you can't modify the values of a "key" line if a "state" that uses it (the items in its foldout area) is "On". All the states that use a systemKey will need to be disabled ("Off") in order for you to modify the values of the "key" manually.
>
> You might want to use the Color Wraps tab and press Toggle Active Sibling to choose a color wraps preset.

Usage:



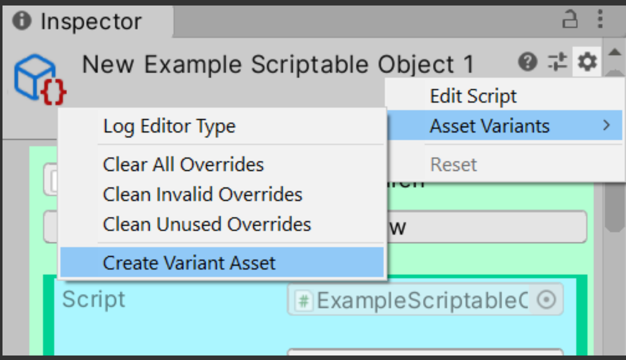
You might want to try out Asset Variants first using the files in /Asset Variants Examples/.

Assigning an asset to the parent field of an asset editor will convert the selected asset/s to variant/s (children) of the parent asset. It will compare the values of the new children's SerializedProperties to those of the assigned parent's in order to create overrides.



The GUI label "Parent" will be hidden when an asset has children, to give space for the Children foldout.

Hold control when pressing an override button to prevent sub-overrides from being removed.



If you want to create a variant for an asset in 2/3 clicks (instead of duplicating the asset then assigning the parent manually), you can use Asset Variants/Create Variant Asset in the inspector context menu for an existing asset. This will also assign parents for editable sub-assets.

To make a duplicate of an asset variant, so that the overrides and parent get copied (Unity doesn't copy an AssetImporter's userData normally), you can use the menu option Assets/Duplicate Asset.

If you want to make use of emulation properties, i.e. DictionaryEmulationProperty, HashSetEmulationProperty, and StringHashSetEmulationProperty, in order to make Arrays/Lists/strings(split by character/s such as ';' or ' ') behave like HashSets or Dictionaries, where the order of the elements doesn't matter, and elements can be added/removed if there is an override for the key, you need to use code like this:

```
[InitializeOnLoadMethod]
private static void Init()
{
    AssetVariants.PropertyFilter.AddEmulationProperty(typeof(Material), new
    StringHashSetEmulationProperty("m_ShaderKeywords", " "), true);

    AssetVariants.PropertyFilter.AddEmulationProperty(typeof(AnimatorOverrid
    eController), new DictionaryEmulationProperty("m_Clips",
    "m_OriginalClip"), true);
}
```

`Assets/Asset Variants/Validate Selected` can be used to force a revert/propagatation of the values, in case something wrong happens.

`AssetVariants.AVUtility` can be useful. These are some methods:

`CreateOverridesFromDifferences()`: it will run the code that normally is run when assigning a new parent to an asset; it will create overrides for the `SerializedProperties` that have different values between it and its parent. This can be used after modifying an asset using code, to almost simulate implicit overriding.

`ValidateRelations()`: with Asset Variants, parent assets have identifiers for their children, and children have an identifier for their parent. If something anomalous occurs to cause disagreements, where an asset thinks another asset is their child, but they are not the father (anymore, or never was), then these relations can get cleared up and fixed with this method. It will prioritize a child’s identifier of its parent, over a parent’s identifiers of its children. It will also remove references to nonexistent children or nonexistent parents. This method gets run automatically at certain times, for example when selecting an asset.

`PropagateChangesToChildren()`: If you modified a parent asset from code, and you didn’t call `OnValuesChanged()`, you probably need to call this, in order to update its children assets.

`CleanOverrides()`: Removes override paths that don't have a `SerializedProperty` in the asset. This can include array elements that no longer exist because the array has been resized to a smaller Length. It can also remove unnecessary overrides for properties that have the same value in the asset and its parent.

`CreateVariant()`: You can call this to duplicate the given asset, and then afterward it will assign the given asset as the parent for the newly created asset. It will do this for each editable sub-asset as well.

`ChangeParent()`: You give this 2 assets/asset paths, and it will set one to be the parent of the other. (This will also automatically create overrides from differences). If you used the path version, it will do this for each editable sub-asset as well (if it can find a `localId` that matches).

`FillRootOverrides()`: This will create overrides for every root level overrideable `SerializedProperty` of the given variant asset. This means none of its parent’s values should be able to propagate to itself.

It’s possible that I overlooked some operation you want to be accessible in the `AVUtility` class, so let me know if there is.

Troubleshooting:

```
#if ASSET_VARIANTS
    using AssetVariants;
#endif
```

Override paths are always stored in Unity's path format, even if they're Odin serialized. Unity's format: `.Array.data[0]` – Odin's format: `.$0`

Odin groups (`.#GroupName`) are not included in odin serialized properties' paths.

If a SerializedProperty is being a problem, if it should be always ignored; if it's not desirable for it to be copied from parent asset to children assets, then you can write something like:

```
[InitializeOnLoadMethod]
private static void Init()
{
    PropertyFilter.IgnorePropertyPath(typeof(Material), "m_SavedProperties");
}
```

If you come across such a property in a Unity/third-party asset type, please do notify me (precisioncats@outlook.com), so I can include it as an `IgnorePropertyPath()` in an update to Asset Variants. Or if you're the developer for your asset, you can write the above code in a block of:

```
#if ASSET_VARIANTS
#endif
```

Such properties include:

- Data generated by the asset, such as large arrays.
- Data that defines the asset as a unique `Object`. This can be hashCodes for example.
- Fields that are co-dependent on other fields; if the asset can be in an invalid state when one field is overridden and copied, but the other field/s are not.

In Raw View you can cause the labels to draw the actual names, the `propertyPath` names, instead of the nice display names, by holding the alt key. You might need to trigger a Repaint by moving your mouse around though.

Adding DEBUG_ASSET_VARIANTS to scripting define symbols might be useful.

You can contact me at
precisioncats@outlook.com

You can change the type of asset that is selectable in the parent field by writing something like:

```
[InitializeOnLoadMethod]
private static void Init()
{
    AVEditor.parentFilterTypes[typeof(Vector42)] = typeof(Vector4224);
    AVEditor.parentFilterTypes[typeof(Vector24)] = typeof(Vector4224);
}
```

You can find out what the replaced and original (if they differ) editor types of an asset are by the context menu option: `Asset Variants/Log Editor Type`. Then you can use code like:

```
EditorReplacement.ERHelper.FindType("UnityEditor.PrefabImporter")
```

to find the `Type` of the editor name that was logged. This is very useful in case the editor type is internal, or if you want to blacklist a type from an optional assembly, without creating a dependency and such to preprocessor it away.

You can contact me at
precisioncats@outlook.com

Some Limitations:

- Layermask can only be overridden as a whole, because the underlying type is an int.

- Quaternions can only be overridden as a whole, or in Raw View as the 4 floats, not as the Euler angles it's drawn as in the inspector.

- The data that can be copied and overridden are only `SerializedProperties` (and `InspectorProperties` in Odin's case). It can't for example copy underlying texture data.

- If you're already using the `userData` of an `AssetImporter`, you need to replace:

```
value = AssetImporter.userData;
```

with:

```
value = AssetImporter.GetUserData(string key, out var onValidateOwnership);
```

(where onValidateOwnership is SharedUserDataUtility.ValidateOwnership),

and you need to replace:

```
AssetImporter.userData = value;
```

with:

```
AssetImporter.SetUserData(string key, string newUserData, ...)
```

That's because Asset Variants also needs to use `userData` to store parent id, children ids, and the overridden property paths.

- Custom material editors are quite likely to have properties drawn in ways incompatible with AVMaterialEditor’s drawer replacement, so you might need to enter Raw View in order to see or remove overridden states.

- If an object is referenced multiple times with Managed References, its properties need overrides in every location or the values will be copied from the parent. Implicit override creation will only occur for the path where you modify a field.

- Managed References in versions before 2021.2 have a maximum depth defined by `ManagedReferencesHelper.maxManagedReferenceDepth`, which defaults to 10. Cyclical references won’t crash or freeze Unity, but every reference to a field up to the `maxManagedReferenceDepth` will need to be overridden (which btw can be done with a single outer override). In 2021.2+ a stack is used to skip if a managed reference has already been referenced by a parent property (i.e. it’s cyclical).

- An override won't be renamed if the property doesn't exist (for example an element in an array that used to have a longer Length), but if an override is left behind then you should be using the context menu `Clean [Unused/Invalid] Overrides`.

- Reordering an `Array/List(/Stack/Queue)` does not move overrides. Overrides are saved by absolute path, with a fixed index into their array. (Except if they're Odin `Dictionaries/HashSets`).

- `OdinPropertyResolvers` can twist the format of the data of a field, and you might need to write something like:

`public class RectRPR : AssetVariants.RawPropertyResolver<Rect> { }` for some type, to have override buttons show up for it.

- Extremely unlikely, but data serialized by Unity can't be copied to data serialized by Odin. This should never matter unless the parent and child don't share the same base class, and both have a field of the exact same name but different `SerializationBackend`.