Texture Array Essentials

The offline documentation can be hard to follow so using the online documentation is recommended which you can find at https://docs.wilschack.dev/texture-array-essentials. You can still use this if you need though, it has all the same information

William Schack

Table of contents

1. Getting Started With TAE

1.1 Introduction

Texture Array Essentials is a tool that allows you to easily create, modify, and use Texture2DArrays through simple menus or custom GUI of your own.

1.2 Tutorial Video

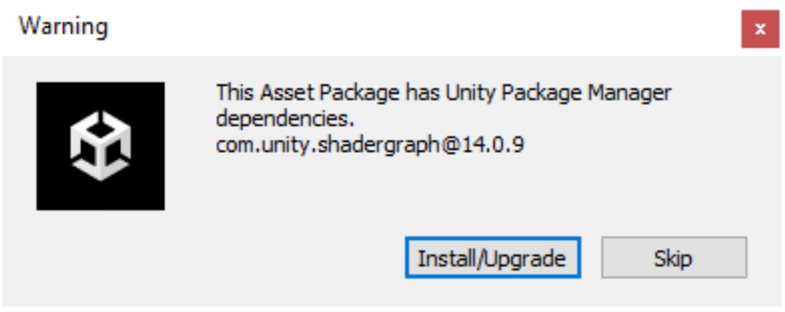
https://youtu.be/xJzasF8RglY

1.3 Contents

1.4 Installation

Install the package through the Package Manager

Using the shader graph functions requires the Shader Graph to be installed and is a dependency for the package. Make sure to click Install/Upgrade when prompted about it.

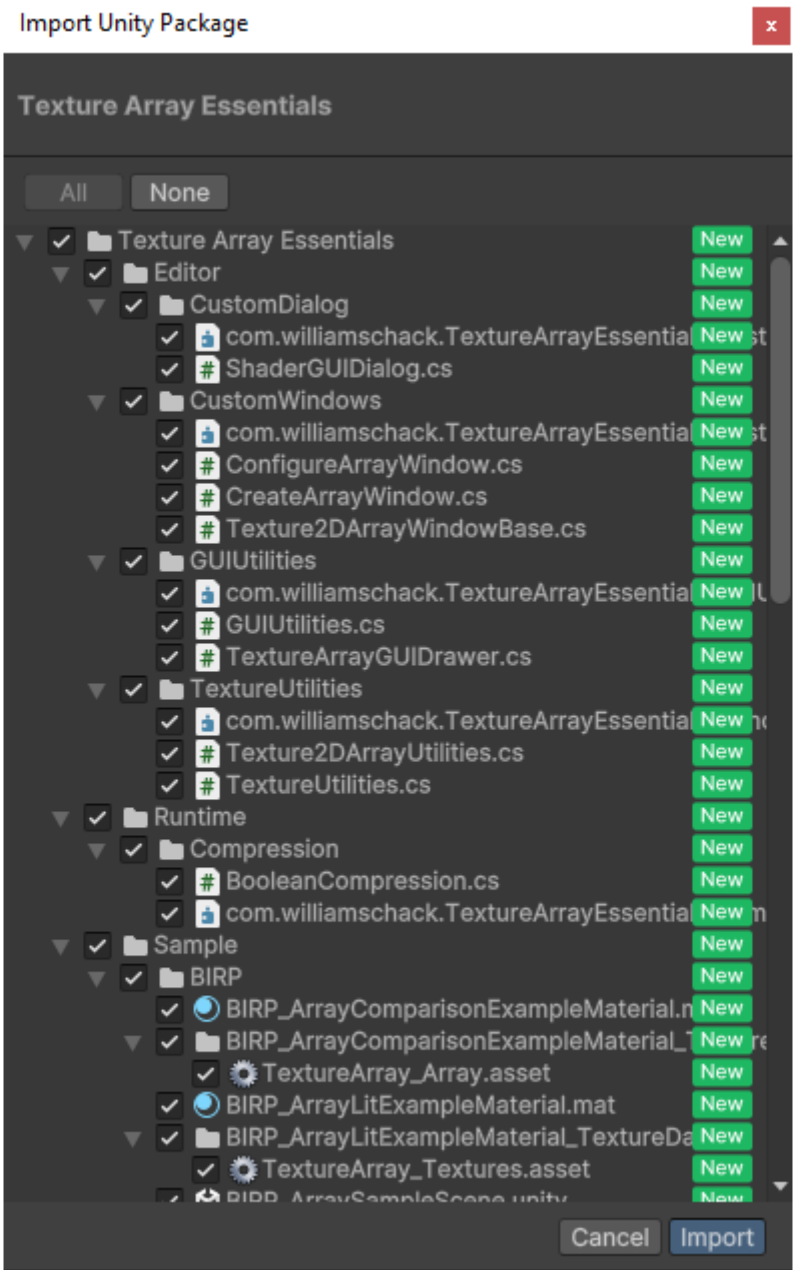

Warning

This Asset Package has Unity Package Manager dependencies.
com.unity.shadergraph@14.0.9

When importing the files, make sure to have all the files selected except for:

- The Sample folder if you do not want the samples
- The Shaders folder if you do not want the shader functions for reading arrays

Once the files are selected, click the import button



```
Import Unity Package                                                    ×
Texture Array Essentials
All   None
▼ ✓ Texture Array Essentials                                      New
  ▼ ✓ Editor                                                      New
    ▼ ✓ CustomDialog                                              New
        ✓ com.williamschack.TextureArrayEssential                 New st
        ✓ ShaderGUIDialog.cs                                      New
    ▼ ✓ CustomWindows                                             New
        ✓ com.williamschack.TextureArrayEssential                 New st
        ✓ ConfigureArrayWindow.cs                                 New
        ✓ CreateArrayWindow.cs                                    New
        ✓ Texture2DArrayWindowBase.cs                             New
    ▼ ✓ GUIUtilities                                              New
        ✓ com.williamschack.TextureArrayEssential                 New U
        ✓ GUIUtilities.cs                                         New
        ✓ TextureArrayGUIDrawer.cs                                New
    ▼ ✓ TextureUtilities                                          New
        ✓ com.williamschack.TextureArrayEssential                 New n
        ✓ Texture2DArrayUtilities.cs                              New
        ✓ TextureUtilities.cs                                     New
  ▼ ✓ Runtime                                                     New
    ▼ ✓ Compression                                               New
        ✓ BooleanCompression.cs                                   New
        ✓ com.williamschack.TextureArrayEssential                 New m
  ▼ ✓ Sample                                                      New
    ▼ ✓ BIRP                                                      New
        ✓ BIRP_ArrayComparisonExampleMaterial.m                   New
      ▼ ✓ BIRP_ArrayComparisonExampleMaterial_T                   New re
          ✓ TextureArray_Array.asset                              New
        ✓ BIRP_ArrayLitExampleMaterial.mat                        New
      ▼ ✓ BIRP_ArrayLitExampleMaterial_TextureDa                  New
          ✓ TextureArray_Textures.asset                           New
        ✓ BIRP_ArraySampleScene.unity                             New
                                                      Cancel   Import
```

1.5 Scripting

For creating your own scripts and GUI using a Texture 2D Array, view the respective Scripting API pages for details on each set of functions and how to implement them into your own code.

Note that all scripts except compression require the Unity Editor to function.

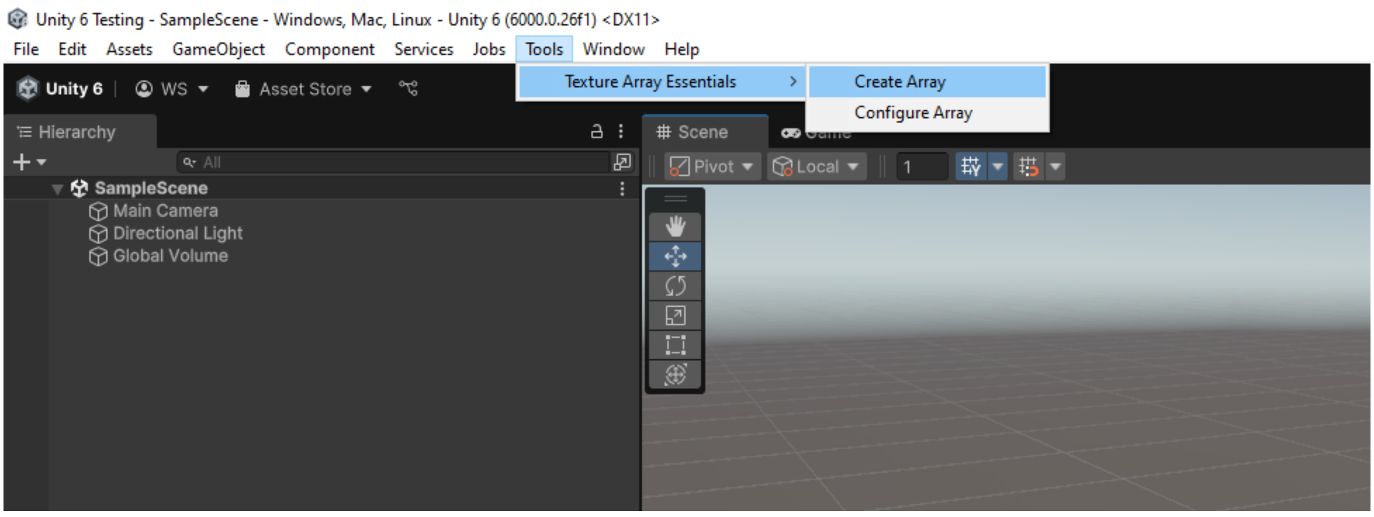
2. Creating/Configuring An Array

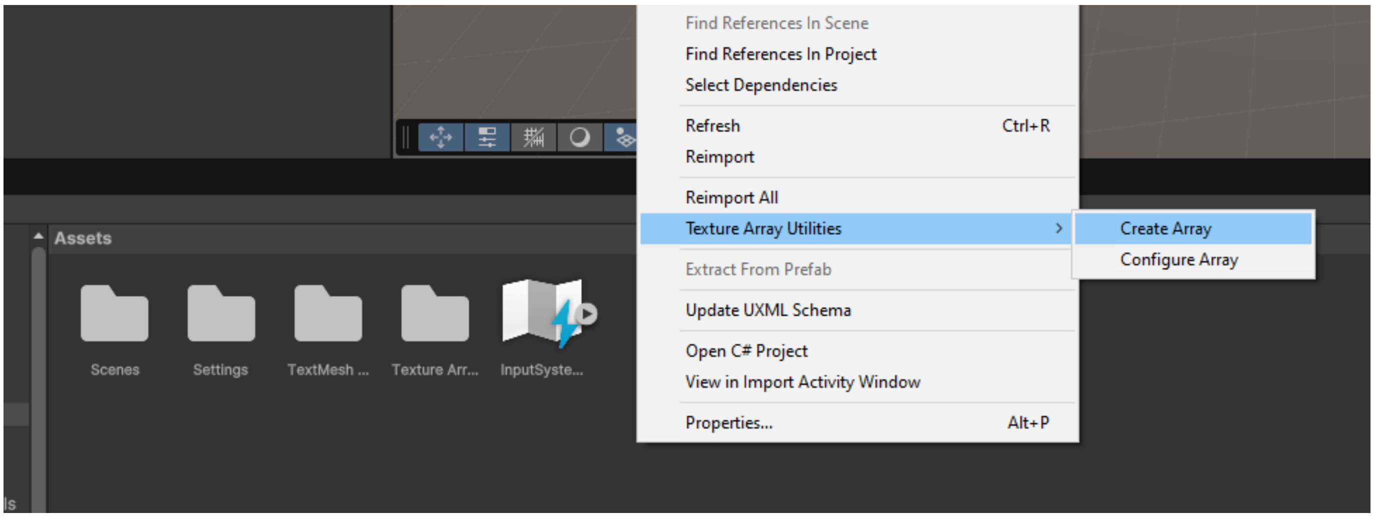
Both the Create and Configure Array windows have the same options with the

2.1 Opening the Creation/Configure Window

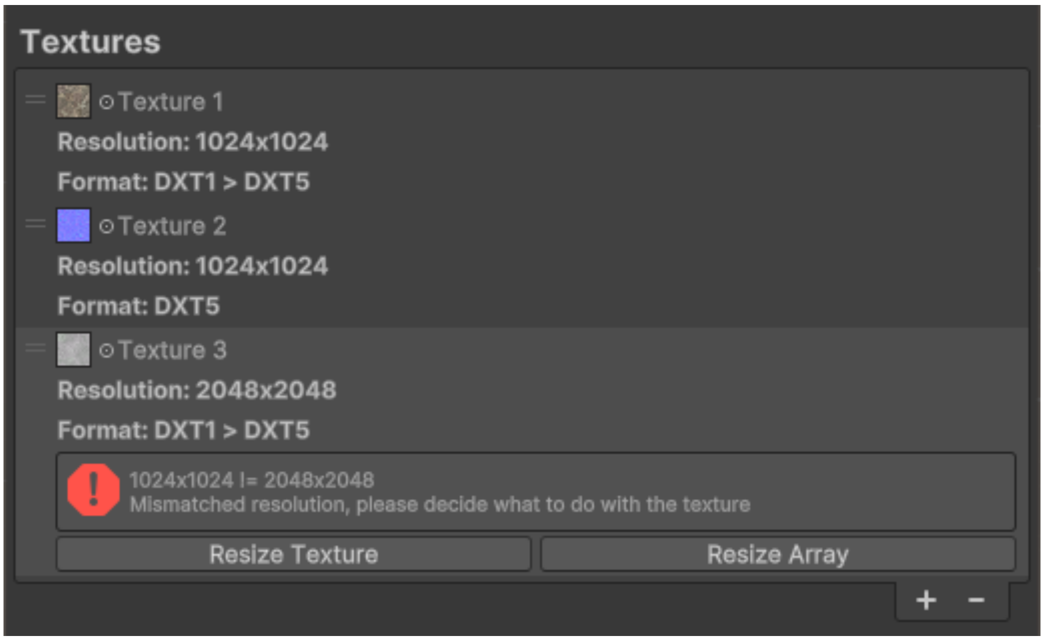
You can open the Create/Configure Array windows through the Tools section in the toolbar or in the context menu in the Project window

You can also open the windows with textures or an array pre-assigned by having textures or an array selected while opening the window





2.2 Modifying Textures



Textures

Texture 1
Resolution: 1024x1024
Format: DXT1 > DXT5

Texture 2
Resolution: 1024x1024
Format: DXT5

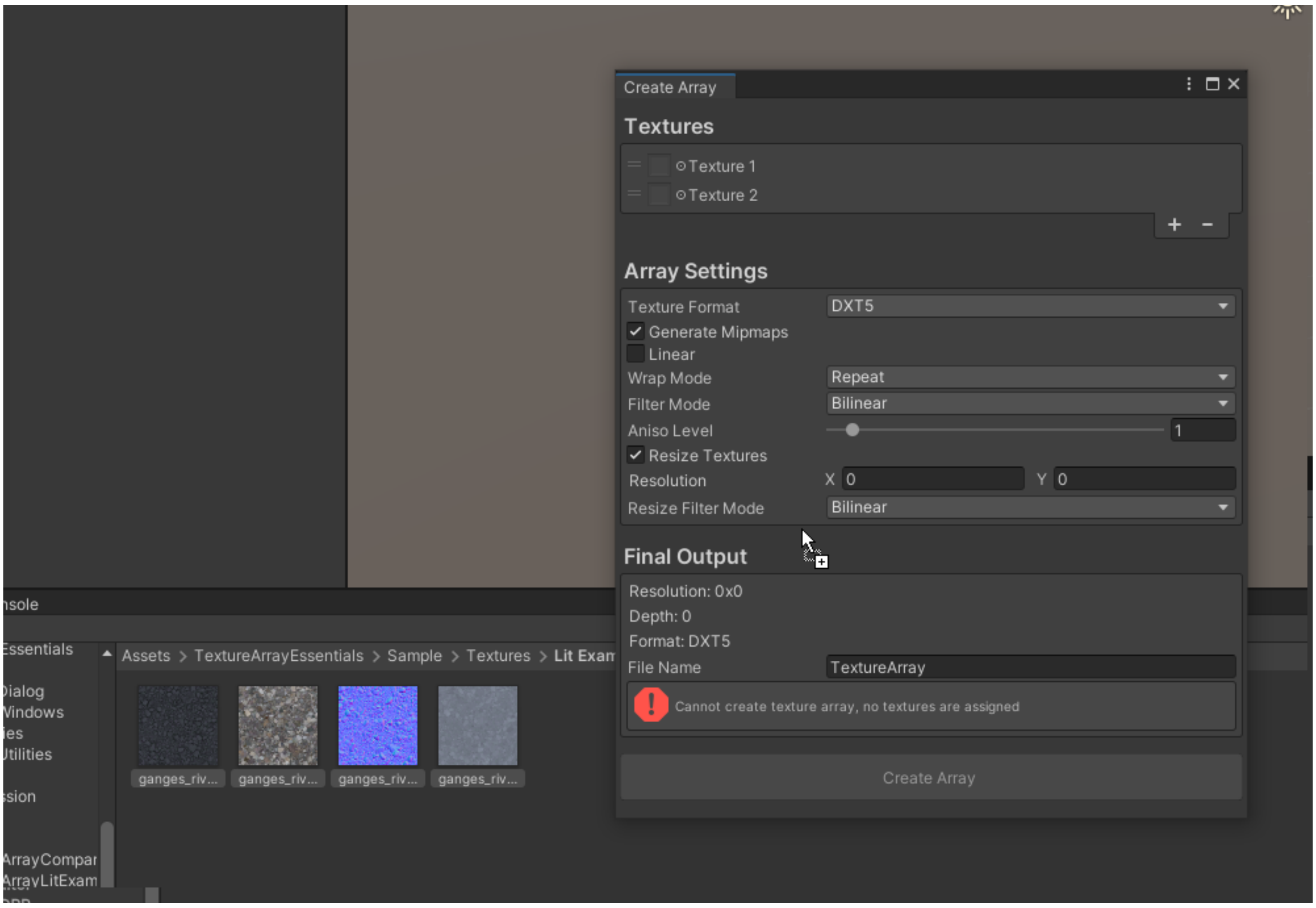
Texture 3
Resolution: 2048x2048
Format: DXT1 > DXT5

1024x1024 != 2048x2048
Mismatched resolution, please decide what to do with the texture

Displays the textures that will be input into the output Texture Array

Textures can be added, removed, and re-arranged with any warnings or errors with an individual texture appearing beneath it.

Textures can be added through the GUI but can also be dragged into the window and will be added to the current textures.



Create Array

Textures

Texture 1
Texture 2

Array Settings

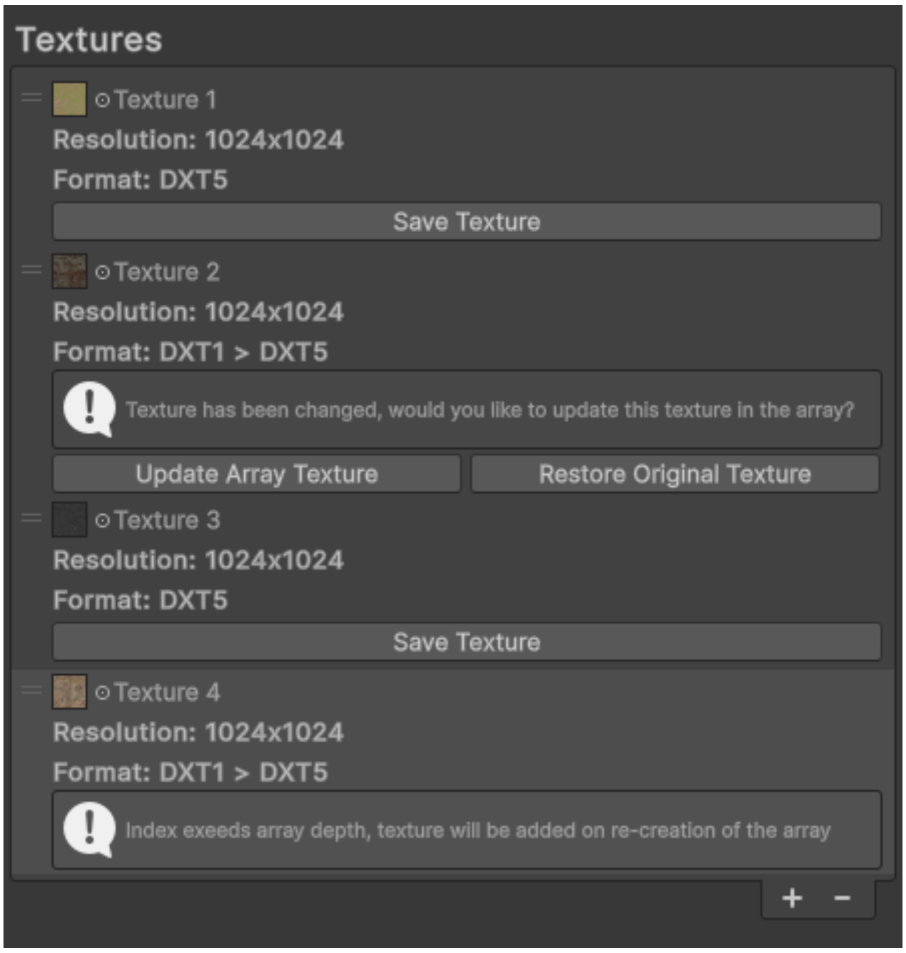
Texture Format: DXT5
✓ Generate Mipmaps
Linear
Wrap Mode: Repeat
Filter Mode: Bilinear
Aniso Level: 1
✓ Resize Textures
Resolution: X 0 Y 0
Resize Filter Mode: Bilinear

Final Output

Resolution: 0x0
Depth: 0
Format: DXT5
File Name: TextureArray

Cannot create texture array, no textures are assigned

The **Configure Array** window has more options when adding textures as you can update individual textures in the array without re-creating it as well as info for when the textures will exceed the original depth of the array.


Textures

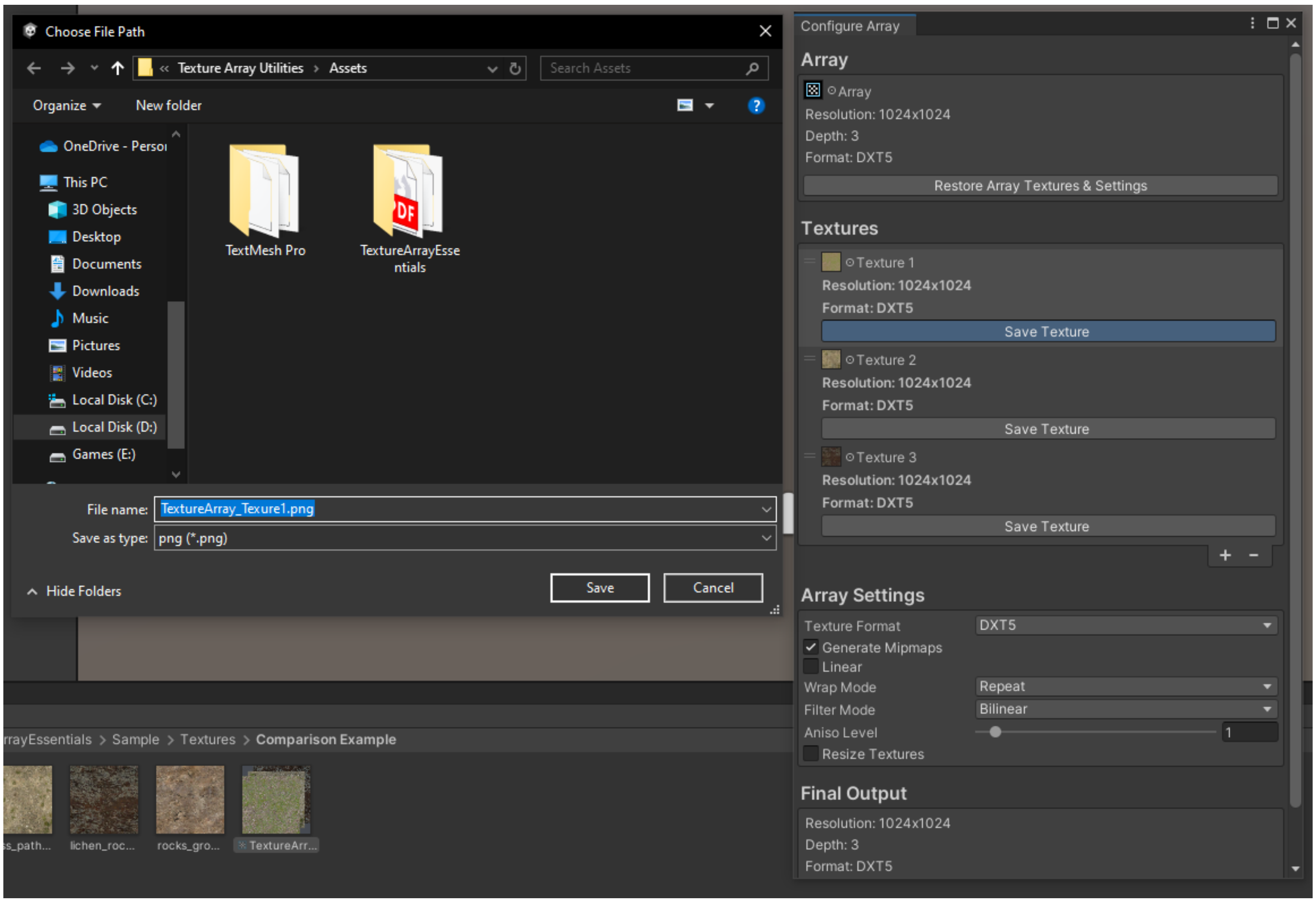
Texture 1
Resolution: 1024x1024
Format: DXT5

Texture 2
Resolution: 1024x1024
Format: DXT1 > DXT5
Texture has been changed, would you like to update this texture in the array?

Texture 3
Resolution: 1024x1024
Format: DXT5

Texture 4
Resolution: 1024x1024
Format: DXT1 > DXT5
Index exeeds array depth, texture will be added on re-creation of the array

In the **Configure Array** window you can also save textures to any folder by clicking the Save Texture button. This is not restricted to the project files and can be saved anywhere on your computer



File name: TextureArray_Texure1.png

Save as type: png (*.png)

Array

Array
Resolution: 1024x1024
Depth: 3
Format: DXT5

Textures

Texture 1
Resolution: 1024x1024
Format: DXT5

Texture 2
Resolution: 1024x1024
Format: DXT5

Texture 3
Resolution: 1024x1024
Format: DXT5

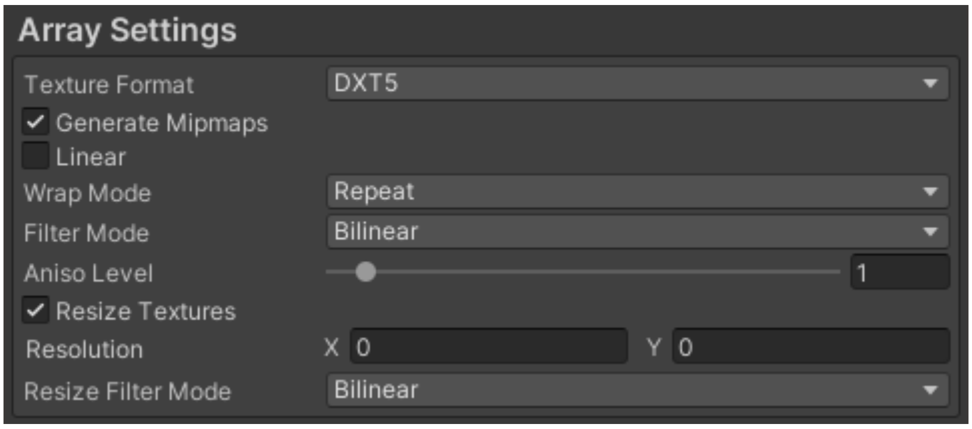
Array Settings

Texture Format: DXT5
✓ Generate Mipmaps
Linear
Wrap Mode: Repeat
Filter Mode: Bilinear
Aniso Level: 1
Resize Textures

Final Output

Resolution: 1024x1024
Depth: 3
Format: DXT5

2.3 Settings



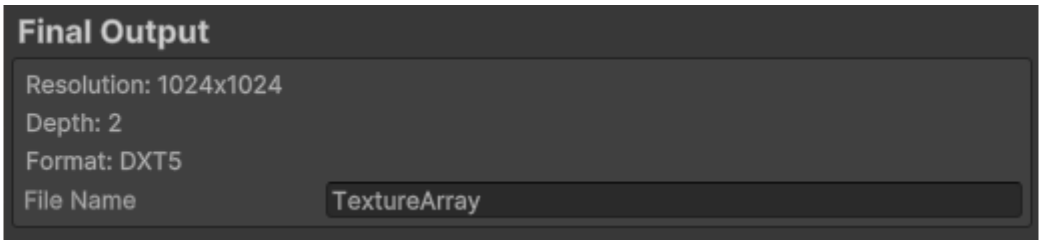
Array Settings

Texture Format: DXT5
✓ Generate Mipmaps
Linear
Wrap Mode: Repeat
Filter Mode: Bilinear
Aniso Level: 1
✓ Resize Textures
Resolution: X 0 Y 0
Resize Filter Mode: Bilinear

The settings determine the output Texture Array. Each setting is listed with a description below

Parameter	Description
Texture Format	The format of the output array
Generate Mipmaps	If mipmaps will be transferred from the textures to the array
Linear	If the output array will be linear Recommended in the Built-In Render Pipeline only when including normal maps **Not Recommended in URP/HDRP as it will result in brighter textures**
Wrap Mode	The Wrap Mode of the output Texture Array
Filter Mode	The Filter Mode of the output Texture Array
Aniso Level	The Aniso Level of the output Texture Array
Resize Textures	If all the inputted textures will be resized to the Array Resolution
Resolution	The resolution that all inputted textures will be resized to if Resize Textures is enabled
Resize Filter Mode	The filter mode used when resizing textures

2.4 Final Output


Final Output
Resolution: 1024x1024
Depth: 2
Format: DXT5
File Name: TextureArray

Displays the details of the output Texture Array with the modifiable file name

Parameter	Description
Resolution	The resolution of the output Texture Array
Depth	How many textures will be included in the output Texture Array
Format	The texture format of the output Texture Array determined by the Compressed setting
File Name	The file name of the output Texture Array **Will be saved to the current folder in the Project Window with the Assets folder as a fallback**

2.5 Creating The Texture Array

In the **Create Array** window, clicking the Create Array button will create a Texture Array with the specified textures and settings in the currently viewed folder in the Project Window with the Assets folder as a fallback.

In the **Configure Array** window, clicking the Re-Create Array button will overwrite the assigned array with the specified textures and settings in the currently viewed folder in the Project Window with the Assets folder as a fallback.

3. Creating a Material Using a Texture Array

The index referring to the inspector position of a texture is named the constant index in the sub-graphs and this page for less confusion

Creating a material with a texture array requires a material, a shader, and a custom GUI.

Accessing the Texture Array requires a bit more work rather than regular textures as a saved array will only have as many textures as assigned in the inspector

3.1 Variables Example

For example, I have 5 texture fields in the inspector but only assign 4, the array will only have 4 textures.

Texture	Assigned	Array Index
1	true	0
2	true	1
3	true	2
4	false	-
5	true	3

In this case if we try to get the 4th texture which we would assume is at index 3, we actually get the 5th texture due to how it is saved. That is why the shader requires an extra variable for each Texture array so it can tell which texture is assigned to what index.

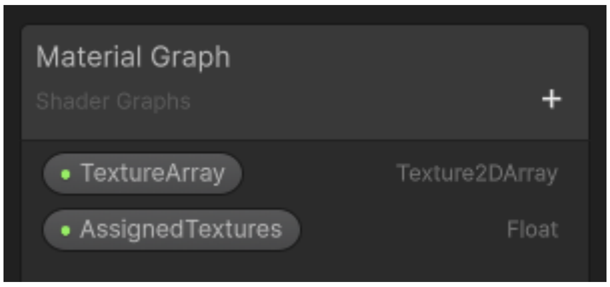
The custom shader functions provide an easy way to get around this so there is no extra work required on your end

3.2 Creating The Shader

I will be using the shader graph for this example example but using a written shader shader should work the same, the sub-graphs used in the shader graph are just functions in the HLSL files you can find at "TextureArrayEssentials/Shaders/HLSL"

Per every texture array in the shader, you need to have two variables:

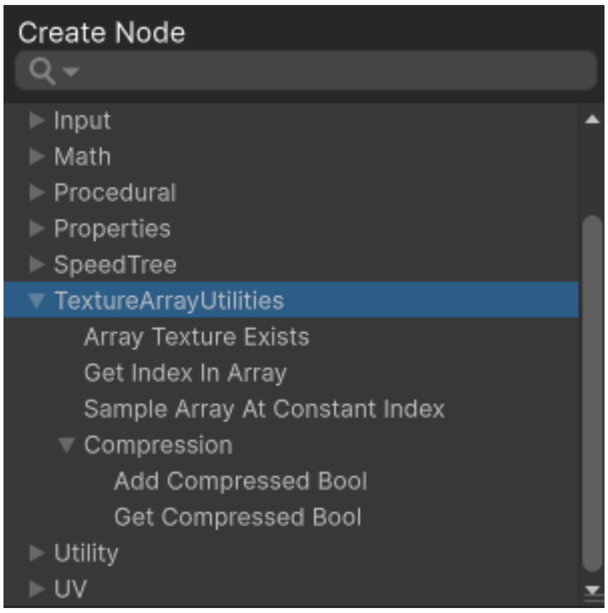
Type	Description
Texture2DArray	The Texture Array
int	Tells the shader and GUI the location of each texture in the array


Material Graph
Shader Graphs

TextureArray	Texture2DArray
AssignedTextures	Float

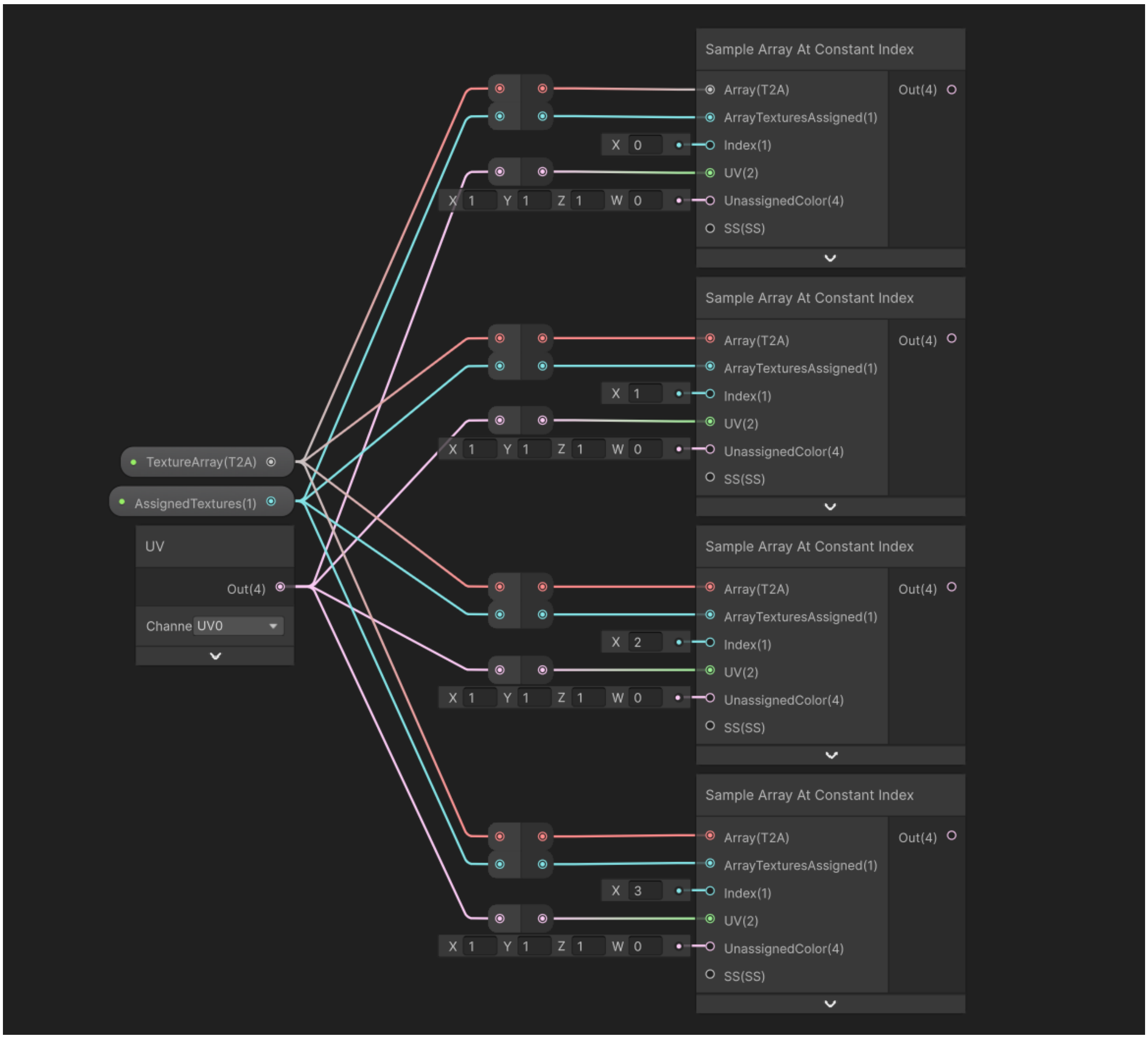
For using the variables there are various sub-graphs to access the array and tell which texture is at what index. Using the texture array with this extra variable also allows you to tell if a texture is assigned in the inspector inside the shader which can be helpful at times.

Function	Returns	Description
Array Texture Exists	bool	Checks if a texture is assigned at the index ***Same as Get Compressed Bool***
Get Index In Array	int	Takes a constant index (inspector index) and gets its index in the Texture Array **Unless you know the texture is assigned, it is a good idea to check Array Texture Exists before using this as it may not return what you expect**
Sample Array At Constant Index	float4	Samples the array at a constant index (inspector index)
Add Compressed Bool	int	Inserts a bool into a compressed set of bools at a given index ***This compression is used by the AssignedTextures***
Get Compressed Bool	bool	Gets a bool value from a set of compressed bools ***This compression is used by the AssignedTextures***



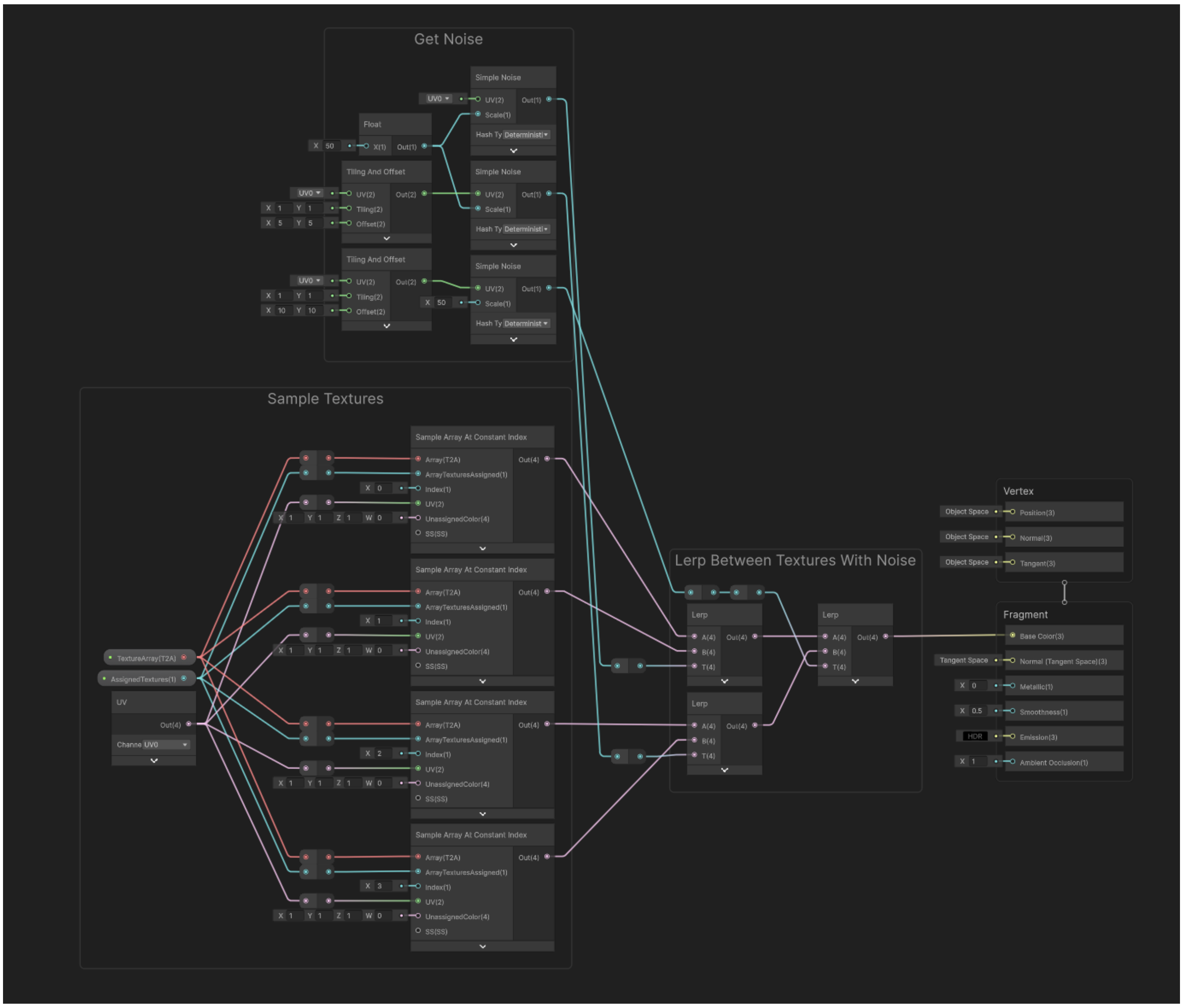
In this example we will be setting up a material with 4 assignable textures, using noise to separate them all.

To get each texture you can use the Sample Array At Constant Index node and input the index for each texture, here being from 0-3



The texture at this point can be used for anything you like with the output of these functions

For this example though, we are using noise to separate the textures so here is the shader, sampling three simple noise functions and lerping the 4 textures with the noise. *(It is a bit messy but more to be used as an example to show how it can be used)*



3.3 Creating a Custom Inspector

For creating a GUI with the shader graph, we will create a class inherited by ShaderGUI

Drawing fields with the Texture2DArray requires a TextureArrayGUIDrawer that must be initialized on Enable of the inspector. This is not a function in the ShaderGUI so you should create one, or at least check for the first call of OnGUI to create the object.

Still following the previous example, we are creating an inspector with 4 assignable textures that will update the Texture Array. Here they are just being drawn as regular textures but there are other functions to draw it along side other fields like a slider or colour field.

For more information on the TextureArrayGUIDrawer, you can find the documentation here

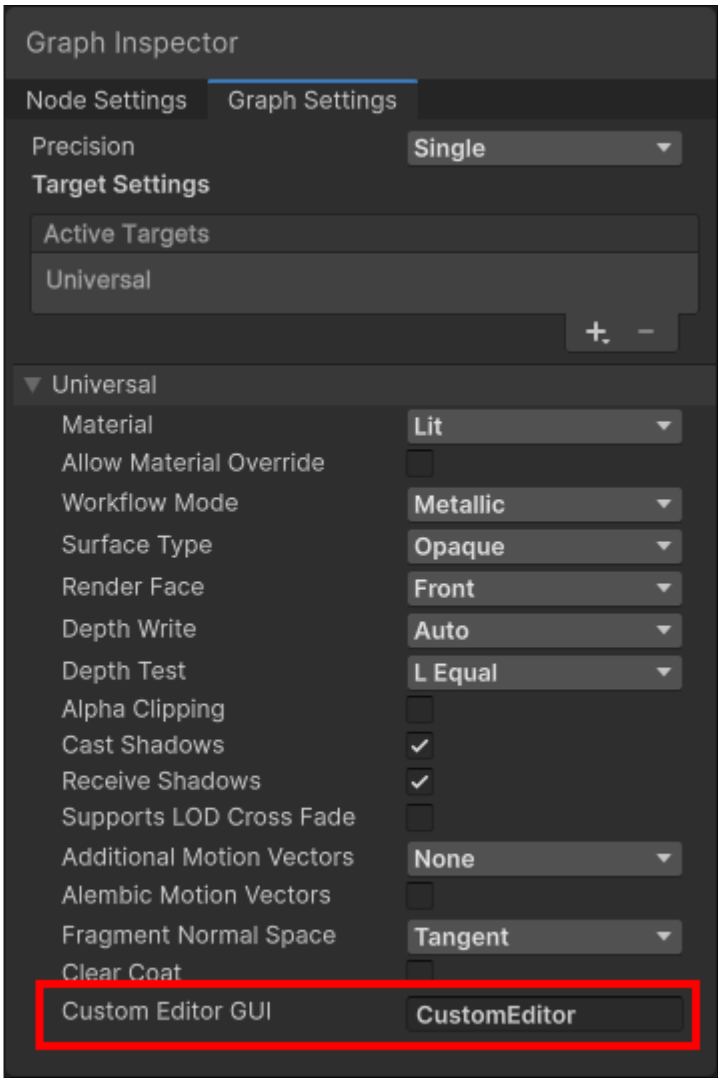
If you want any extra GUI functions, GUIUtilities has more functions and helpers for drawing various elements without the need for MaterialProperties. You can find the documentation here

```
#if UNITY_EDITOR
using UnityEditor;
using TextureArrayEssentials.GUIUtilities;

public class CustomEditor : ShaderGUI
{
    TextureArrayGUIDrawer _arrayDrawer;

    // ShaderGUI doesnt have an OnEnable function, using this instead
    private bool _firstSetup = true;
    private void OnEnable(MaterialProperty[] properties)
    {
        // Get material properties
        MaterialProperty textureArrayProperty = FindProperty("_TextureArray", properties);
        MaterialProperty assignedTexturesProperty = FindProperty("_AssignedTextures", properties);

        // Create the array drawer
        // In this example there are 4 textures that can be assigned
        _arrayDrawer = new TextureArrayGUIDrawer(textureArrayProperty, assignedTexturesProperty, 4);
    }

    public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
    {
        // OnEnable if first call
        if (_firstSetup) {
            OnEnable(properties);
            _firstSetup = false;
        }

        // Draw textures
        _arrayDrawer.DrawTextures();

        // Can be drawn seperately using:
        //_arrayDrawer.DrawTexture(0);
        //_arrayDrawer.DrawTexture(1);
        //_arrayDrawer.DrawTexture(2);
        //_arrayDrawer.DrawTexture(3);
    }
}

#endif
```

3.4 Assigning the Custom Inspector

You can assign the custom inspector to a Shader Graph through the GraphSettings in the Custom Editor GUI Field

This has the naming convention of **"namespace.classname"** and in this case since there is no namespace, we just put CustomEditor, the name of the class


Graph Inspector

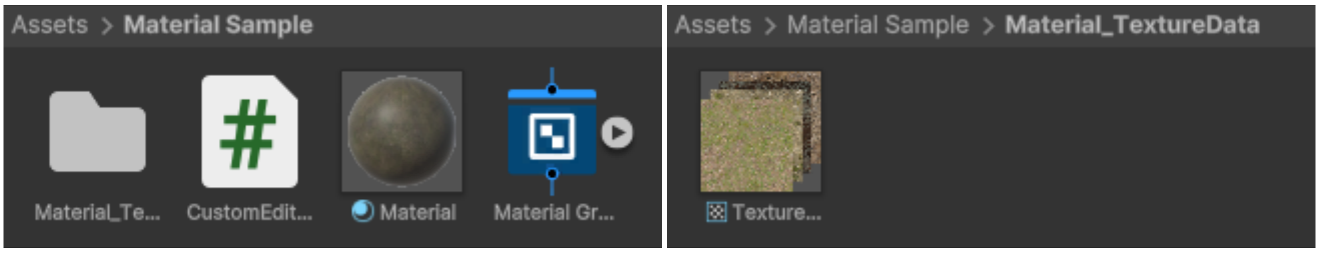
Node Settings | Graph Settings

Precision	Single
Target Settings	
Active Targets	
Universal	
Universal	
Material	Lit
Allow Material Override	☐
Workflow Mode	Metallic
Surface Type	Opaque
Render Face	Front
Depth Write	Auto
Depth Test	L Equal
Alpha Clipping	☐
Cast Shadows	✓
Receive Shadows	✓
Supports LOD Cross Fade	☐
Additional Motion Vectors	None
Alembic Motion Vectors	☐
Fragment Normal Space	Tangent
Clear Coat	☐
Custom Editor GUI	CustomEditor

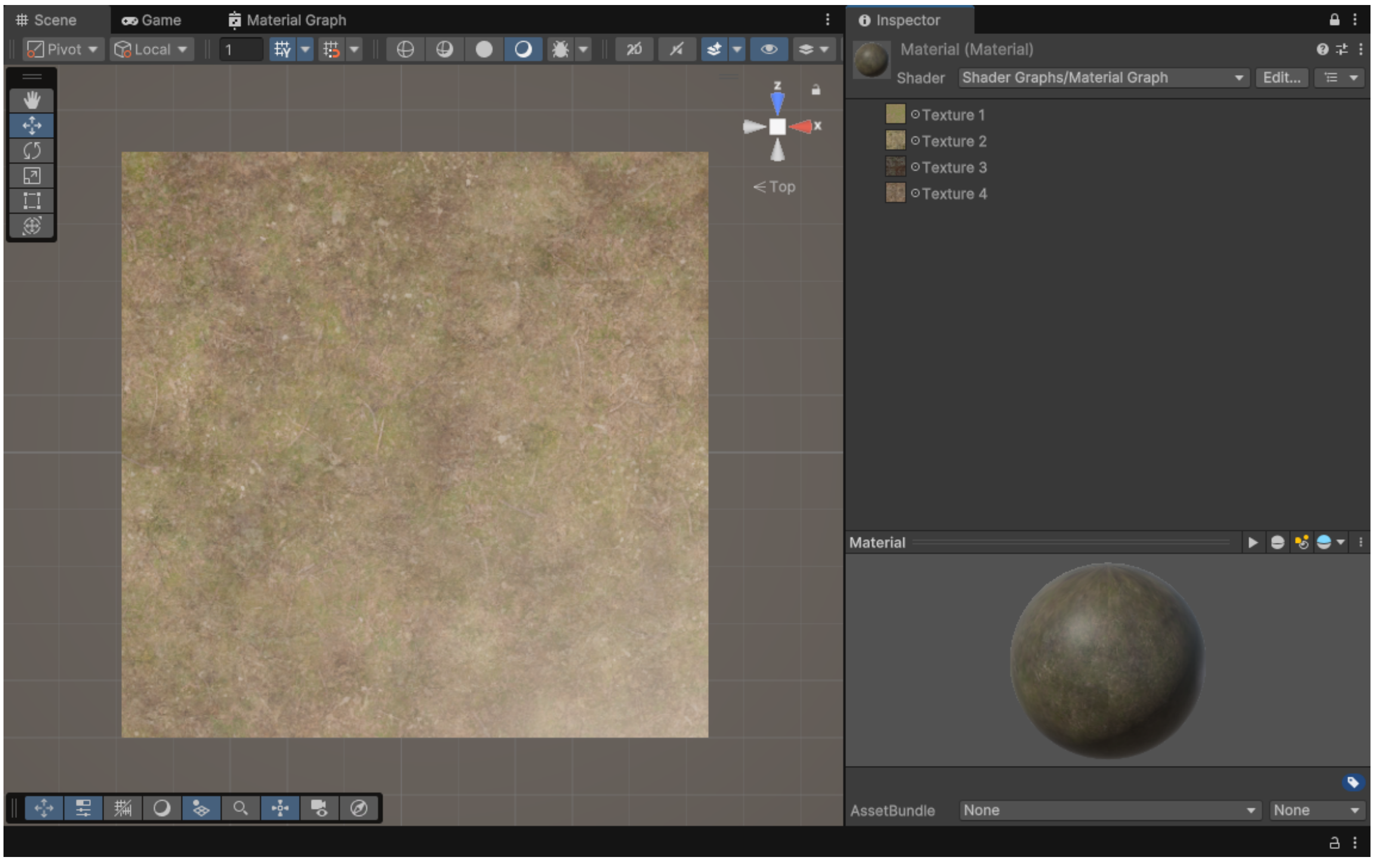
3.5 Using the Material

Now the material is done! You can use it like any other material and the Texture array will handle itself.

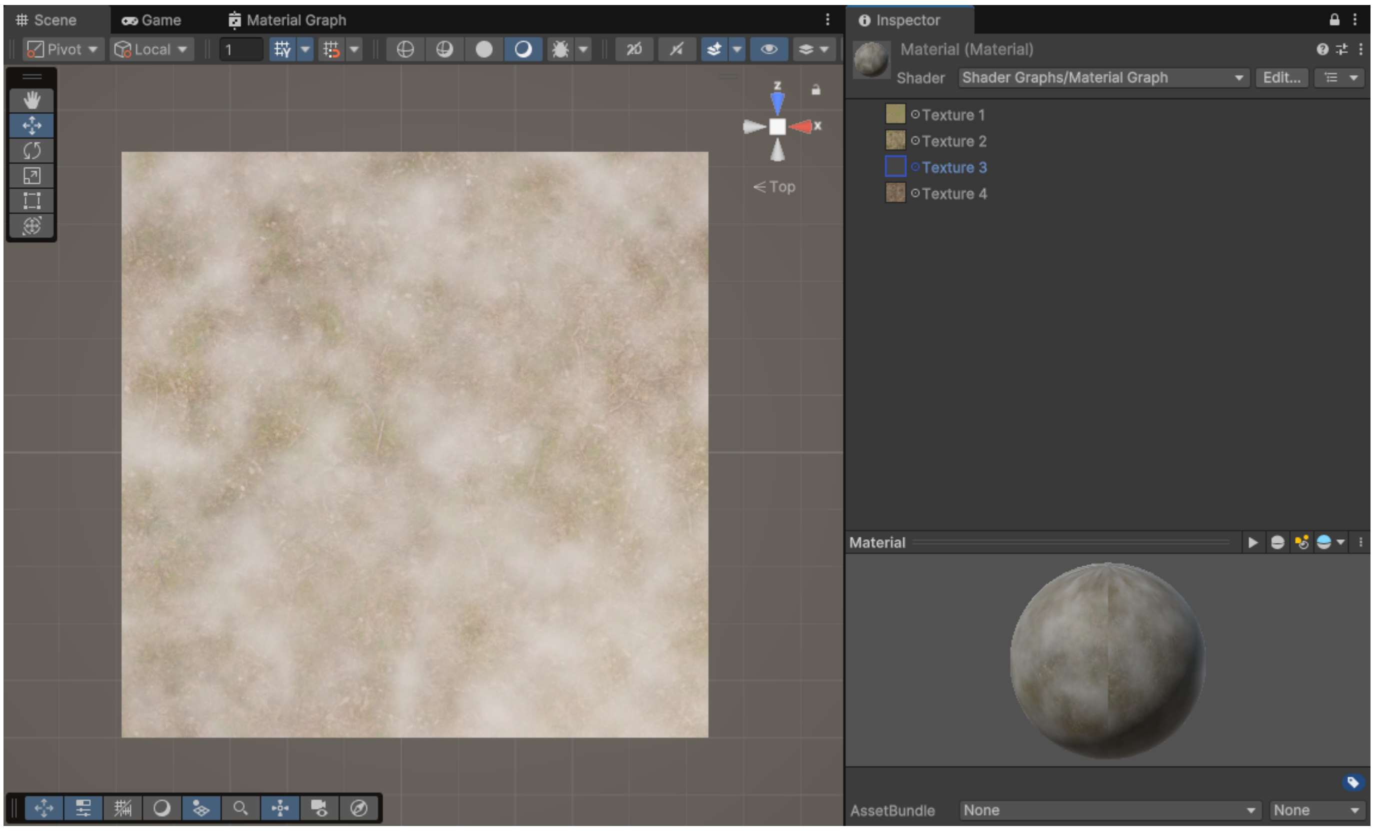
Since the Texture2DArray is its own asset, the material has its own folder that holds each texture array along side it. The folder is only used on creation of a new array and can be moved to any location as the GUI gets the path from the assigned array.



In our example, the inspector looks like this and each texture can be assigned like any other texture field.



You can see in the scene view that an unassigned texture will default to the colour white. This can be changed in the shader graph with the UnassignedColor value for each Sample Array At Constant Index node



4. Scripting API

4.1 Texture2DArrayUtilities

4.1.1 Description

`Unity Editor Only`

Various helper functions for Texture2DArrays including:

- Creating a Texture Array
- Modifying a Texture Array
- Retrieving textures from a Texture Array

4.1.2 CreateArray

Declaration

`public static Texture2DArray CreateArray(Texture2D[] textures, bool compressed = true, bool transferMipmaps = true, bool linear = false)`

Parameters

Parameter	Description
textures	The textures that will be in the array
textureFormat	The texture format of the output array
transferMipmaps	Default: **true** If mipmaps will be transferred from the textures to the array
linear	Default: **false** If the output array will be linear Recommended in the Built-In Render Pipeline only when including normal maps **Not Recommended in URP/HDRP as it will result in brighter textures**

Returns

A Texture2DArray with the inputted textures and settings

Description

Creates a new Texture2DArray with the given textures taking into account unsupported formats

Resolution will be the resolution of the first input texture

Texture will be skipped if it is not the same resolution of the first

Example of creating an array and saving it to the Assets folder

```
private void OnGUI()
{
    // Replace with your textures
    Texture2D[] textures;

    // Button that creates the array
    // In this example the array will have the default inputs
    if(GUILayout.Button("Create Array")) {
        // Create array with preset textures
        Texture2DArray array = Texture2DArrayUtilities.CreateArray(textures);

        // Save array to the Assets folder
        AssetDatabase.CreateAsset(array, "Assets/TextureArray.asset");
    }

}
```

4.1.3 CreateArrayAutoResize

Declaration

`public static Texture2DArray CreateArrayAutoResize(Texture2D[] textures, bool compressed = true, bool transferMipmaps = true, bool linear = false)`

Parameters

Parameter	Description
textures	The textures that will be in the array
textureFormat	The texture format of the output array
transferMipmaps	Default: **true** If mipmaps will be transferred from the textures to the array
linear	Default: **false** If the output array will be linear Recommended in the Built-In Render Pipeline only when including normal maps **Not Recommended in URP/HDRP as it will result in brighter textures**

Returns

A Texture2DArray with the inputted textures and settings

Description

Creates a new Texture2DArray with the given textures taking into account unsupported formats

Resolution will be the resolution of the first input texture

Automatically resizes all textures to the resolution of the first

Example of creating an array and saving it to the Assets folder

```
private void OnGUI()
{
    // Replace with your textures
    // In this example imagine the second texture is a different resolution to the first
    // In this case the second texture will be resized to the resolution of the first
    Texture2D[] textures;

    // Button that creates the array
    // In this example the array will have the default inputs
    if(GUILayout.Button("Create Array")) {
        // Create array with preset textures
        // This will resize all textures to the resolution of the first if they are not already
        Texture2DArray array = Texture2DArrayUtilities.CreateArrayAutoResize(textures);

        // Save array to the Assets folder
        AssetDatabase.CreateAsset(array, "Assets/TextureArray.asset");
    }

}
```

4.1.4 CreateArrayUserInput

Declaration

```
public static Texture2DArray CreateArrayUserInput(Texture2D[] textures, int[] autoResizeIndexes = null, bool compressed = true,
bool transferMipmaps = true, bool linear = false)
```

Parameters

Parameter	Description
textures	The textures that will be in the array
textureFormat	The texture format of the output array
autoResizeIndexes	Default: **null** Automatically resizes all textures at these indexes if required instead of showing a prompt to the user Ex. If this is { 0, 2 }, the textures at index 0 & 2 will be automatically resized
transferMipmaps	Default: **true** If mipmaps will be transferred from the textures to the array
linear	Default: **false** If the output array will be linear Recommended in the Built-In Render Pipeline only when including normal maps **Not Recommended in URP/HDRP as it will result in brighter textures**

Returns

A Texture2DArray with the inputted textures and settings

Description

Creates a new Texture2DArray with the given textures taking into account unsupported formats

Resolution will be the resolution of the first input texture

Checks for user input to decide whether to resize the texture or array if a texture is at a different resolution to the array

Example of creating an array and saving it to the Assets folder

```
private void OnGUI()
{
    // Replace with your textures
    Texture2D[] textures;

    // In this example the textures at index 0 and 2 will automatically be resized to the resolution of the first
    // Any other texture in the array will ask for user input
    int[] autoResizeIndexes = { 1, 2 };

    // Button that creates the array
    // In this example the array will have the default inputs
    if(GUILayout.Button("Create Array")) {
        // Create array with preset textures
        // This will resize any textures with an index in the autoResizeIndexes to the resolution of the first
        //  Any other texture in the array will ask for user input if it requires resizing
        Texture2DArray array = Texture2DArrayUtilities.CreateArrayUserInput(textures, autoResizeIndexes);

        // Save array to the Assets folder
        AssetDatabase.CreateAsset(array, "Assets/TextureArray.asset");
    }

}
```

4.1.5 UpdateTexture

Declaration

`public static Texture2DArray UpdateTexture(Texture2DArray array, Texture2D texture, int index, bool transferMipmaps = true)`

Parameters

Parameter	Description
array	The array that will be updated
texture	The texture that will be inserted into the array
index	The index which the texture will overwrite
transferMipmaps	If mipmaps will be transferred from the texture to the array if possible

Returns

The modified array with the texture inserted

Description

Overwrites the texture in the input array at a given index to the input texture

Uses GPU functions to speed up operations

Update will not be applied if the texture resolution is not the same as the array

Example of inserting a texture into a texture array

```
private void OnGUI()
{
    // Replace with your texture and array
    Texture2DArray array;
    Texture2D texture;

    // Button that updates the array
    if(GUILayout.Button("Update Array")) {
        // Update array with new texture
        // In this example the texture will be inserted into index 0
        array = Texture2DArrayUtilities.UpdateTexture(array, texture, 0);
    }

}
```

4.1.6 UpdateTextureAutoResize

Declaration

`public static Texture2DArray UpdateTextureAutoResize(Texture2DArray array, Texture2D texture, int index, bool transferMipmaps = true)`

Parameters

Parameter	Description
array	The array that will be updated
texture	The texture that will be inserted into the array
index	The index which the texture will overwrite
transferMipmaps	Default: **true** If mipmaps will be transferred from the texture to the array if possible

Returns

The modified array with the texture inserted

Description

Overwrites the texture in the input array at a given index to the input texture

Automatically resizes the input texture to the array size if it is a different resolution

Example of inserting a texture into a texture array

```
private void OnGUI()
{
    // Replace with your texture and array
    Texture2DArray array;
    Texture2D texture;

    // Button that updates the array
    if(GUILayout.Button("Update Array")) {
        // Update array with new texture
        // In this example the texture will be inserted into index 0
        // This will resize the texture to the array resolution if it is not already
        array = Texture2DArrayUtilities.UpdateTextureAutoResize(array, texture, 0);
    }

}
```

4.1.7 UpdateTextureUserInput

Declaration

`public static (Texture2DArray, bool) UpdateTextureUserInput(Texture2DArray array, Texture2D texture, int index, bool transferMipmaps = true)`

Parameters

Parameter	Description
array	The array that will be updated
texture	The texture that will be inserted into the array
index	The index which the texture will overwrite
transferMipmaps	Default: **true** If mipmaps will be transferred from the texture to the array if possible

Returns

Item1: Modified Array, Item2: User Cancelled

Description

Overwrites the texture in the input array at a given index to the input texture

Waits for the user to input the outcome when a texture is a different resolution to the array

Example of inserting a texture into a texture array

```
private void OnGUI()
{
    // Replace with your texture and array
    Texture2DArray array;
    Texture2D texture;

    // Button that updates the array
    if(GUILayout.Button("Update Array")) {
        // Update array with new texture
        // In this example the texture will be inserted into index 0
        // This will wait for user input if the texture requires resizing
        array = Texture2DArrayUtilities.UpdateTextureUserInput(array, texture, 0);
    }

}
```

4.1.8 GetTexture

Declaration

`public static Texture2D GetTexture(Texture2DArray array, int index)`

Parameters

Parameter	Description
array	The array that will be searched
index	The index for which texture will be returned

Returns

The texture in the array at the specified index

Description

Retrieves the texture in an input array at a given index

Example of retrieving a texture from a texture array

```
private void Function()
{
    // Replace with your array
    Texture2DArray array;

    // This will retrieve a texture from the array
    // In this example the texture retrieved is at index 0
    Texture2D texture = Texture2DArrayUtilities.GetTexture(array, 0);
}
```

4.1.9 GetTextures

Declaration

`public static Texture2D[] GetTextures(Texture2DArray array)`

Parameters

Parameter	Description
array	The array that will be searched

Returns

An array with all the textures from the given array

Description

Retrieves all the textures in an input array

Example of retrieving a texture from a texture array

```
private void Function()
{
    // Replace with your array
    Texture2DArray array;

    // This will retrieve all the textures from the array
    Texture2D[] textures = Texture2DArrayUtilities.GetTextures(array);
}
```

4.1.10 ResizeArrayTextures

Declaration

`public static Texture2DArray ResizeArrayTextures(Texture2DArray array, int newWidth, int newHeight, FilterMode resizeFilterMode = FilterMode.Bilinear)`

Parameters

Parameter	Description
array	The array that will be searched
newWidth	The width of the output textures
newHeight	The height of the output textures
resizeFilterMode	Default: **FilterMode.Bilinear** The filter mode used when resizing the textures

Returns

The modified array with all the textures at the new resolution

Description

Scales all textures in the array to the new specified resolution

Skips textures already at the new resolution

Example of resizing an array through GUI inputs

```
private void OnGUI()
{
    // Replace with your array
    Texture2DArray array;

    // Button to resize array
    if(GUILayout.Button("Resize Array")) {
        // This will resize the array and all textures in it to the specified resolution
        // In this example the array will be resized to 2048x2048 using a Bilinear filter by default
        array = ResizeArrayTextures(array, 2048, 2048);
    }
}
```

4.2 TextureUtilities

4.2.1 Description

`Unity Editor Only`

Various helper functions for modifying textures

4.2.2 ResizeTexture

Declaration

`public static Texture2D ResizeTexture(Texture2D texture, int newWidth, int newHeight, FilterMode filterMode = FilterMode.Bilinear)`

Parameters

Parameter	Description
texture	The texture that will be resized
newWidth	The width of the output texture
newHeight	The height of the output texture
filterMode	Default: **FilterMode.Bilinear** The filter mode used when resizing the texture

Returns

The texture resized to the input resolution

Description

Scales the input texture to the desired resolution

Does not modify the original texture, returns a new resized one

```
private void Function()
{
    // Replace with your texture you want to resize
    Texture2D textureToResize;

    // Resize the texture to a specific resolution
    // In this example it will be resized to 2048x2048 with a Bilinear filter mode
    // This does not modify the original texture but returns a new one with the new resolution
    Texture2D resizedTexture = TextureUtilities.ResizeTexture(textureToResize, 2048, 2048, FilterMode.Bilinear);
}
```

4.2.3 SetReadable

Declaration

`public static void SetReadable(Texture2D texture, bool logChanges = false)`

Parameters

Parameter	Description
texture	The texture that will set to readable
logChanges	Default: **false** Debugs if the texture is set to readable if it is not already

Description

Re-Imports the input texture as readable if necessary

```
private void Function()
{
    // Replace with your texture you want to read from
    Texture2D texture;

    // Sets the texture to readable if it is not already
    // Modifies the original texture
    TextureUtilities.SetReadable(texture);

    // No return value so use the texture input into the function here
}
```

Declaration

`public static void SetReadable(Texture2D[] textures, bool logChanges = false)`

Parameters

Parameter	Description
textures	The array of textures that will set to readable
logChanges	Default: **false** Debugs if the texture is set to readable if it is not already

Description

Re-Imports the input textures as readable if necessary

```
private void Function()
{
    // Replace with your textures you want to read from
    Texture2D[] textures;

    // Sets the textures to readable if they are not already
    // Modifies the original textures
    TextureUtilities.SetReadable(textures);

    // No return value so use the textures input into the function here
}
```

4.2.4 ConvertFromCompressedNormal

Declaration

`public static Texture2D ConvertFromCompressedNormal(Texture2D texture)`

Parameters

Parameter	Description
texture	The texture that will be converted

Returns

The texture converted to an uncompressed normal

Description

Converts a unity compressed normal map back to an uncompressed one, from red to blue essentially

Does not modify the original texture, returns a new converted one

```
private void Function()
{
    // Replace with your compressed normal
    Texture2D compressedNormal;

    // Convert the unity compressed normal to a regular normal texture
    // This does not modify the original texture but returns a new one with the uncompressed normal
    Texture2D uncompressedNormal = TextureUtilities.ConvertFromCompressedNormal(compressedNormal);
}
```

4.3 GUIUtilities

4.3.1 GUIUtilities.TextureArrayGUIDrawer

Description

`Unity Editor Only`

Allows drawing textures stored in a Texture2DArray to the GUI as well as functions for reading and deleting the array

Automatically creates and manages its own array when textures are modified

Includes texture fields accompanied by:

- Sliders
- Int/Float fields
- Color field

This is not a static class and an object must be created OnEnable of the inspector. Textures can be drawn by calling the object

Each texture array requires two variables in the shader

- **Texture Array (Texture2DArray)**
- **Assigned Textures (int)**

Example of using a TextureArrayGUIDrawer

```
using UnityEngine;
using TextureArrayUtilities.GUIUtilities;

#if UNITY_EDITOR
using UnityEditor;

public class ArrayGUIExample : ShaderGUI
{
    // The drawer can be used multiple times, once for each array
    private TextureArrayGUIDrawer _arrayDrawerSection1;
    private TextureArrayGUIDrawer _arrayDrawerSection2;

    // ShaderGUI doesnt have an OnEnable function, using this instead
    private bool _firstSetup = true;

    private void OnEnable(MaterialEditor materialEditor, MaterialProperty[] properties)
    {
        // Property names are the references from the shader
        // Replace them with your own
        MaterialProperty arraySection1Property = FindProperty("_ArraySection1", properties);
        MaterialProperty arraySection2Property = FindProperty("_ArraySection2", properties);
        MaterialProperty assignedTexturesSection1Property = FindProperty("_AssignedTexturesSection1", properties);
        MaterialProperty assignedTexturesSection2Property = FindProperty("_AssignedTexturesSection2", properties);

        // Texture amount should be changed to the max amount of textures you want in your array
        // In this example we will use a max of 4
        _arrayDrawerSection1 = new TextureArrayGUIDrawer(arraySection1Property, assignedTexturesSection1Property, 4);
        _arrayDrawerSection2 = new TextureArrayGUIDrawer(arraySection2Property, assignedTexturesSection2Property, 4);

        // Settings of the arrays can be modified by these variables
        // Most of these settings will only take effect when the array is created
        // Array is not created on initialization, only when needed so these can be modified afterwards
        _arrayDrawerSection1.TextureFormat = TextureFormat.DXT5;
        _arrayDrawerSection1.TransferMipmaps = true;
        _arrayDrawerSection1.ArrayLinear = false;

        _arrayDrawerSection2.TextureFormat = TextureFormat.DXT5;
        _arrayDrawerSection2.TransferMipmaps = true;
        _arrayDrawerSection2.ArrayLinear = false;
    }

    public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
    {
        // OnEnable if first call
        if (_firstSetup) {
            OnEnable(materialEditor, properties);
            _firstSetup = false;
        }

        // Example for drawing all the textures in both arrays
        // An array of GUIContent can be assigned for the label and tooltip of each field
        _arrayDrawerSection1.DrawTextures();
        GUILayout.Space(10); // Adds space in between arrays
        _arrayDrawerSection2.DrawTextures();
    }
}

#endif
```

TextureArrayGUIDrawer

DECLARATION

`public TextureArrayGUIDrawer(MaterialProperty arrayProperty, MaterialProperty assignedTexturesProperty, int textureCount, string fileName = null)`

PARAMETERS

Parameter	Description
arrayProperty	The material property for the Array (Texture2DArray)
assignedTexturesProperty	The material property for the Assigned Textures (float)
textureCount	The max amount of textures this array will hold
fileName	Default: **Null** The filename of the Texture2DArray asset stored in a folder accompanying the material Used only on creation of the asset. Array is retrieved from the material afterwards

DESCRIPTION

Create a TextureArrayGUIDrawer using the Array and Assigned Textures properties

The Texture2DArray asset will be stored in a folder accompanying the material. Can be moved after creation

```
TextureArrayGUIDrawer _arrayDrawer;

// ShaderGUI doesnt have an OnEnable function, using this instead
private bool _firstSetup = true;

private void OnEnable(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Property names are the references from the shader
    // Replace them with your own
    MaterialProperty arrayProperty = FindProperty("_Array", properties);
    MaterialProperty assignedTexturesProperty = FindProperty("_AssignedTextures", properties);

    // Create an array drawer
    // Texture amount should be changed to the max amount of textures you want in your array
    // In this example we will use a max of 4
    _arrayDrawer = new TextureArrayGUIDrawer(arrayProperty, assignedTexturesProperty, 4);

    // Settings of the arrays can be modified by these variables
    // Most of these settings will only take effect when the array is created
    // Array is not created on initialization, only when needed so these can be modified afterwards
    _arrayDrawerSection1.TextureFormat = TextureFormat.DXT5;
    _arrayDrawerSection1.TransferMipmaps = true;
    _arrayDrawerSection1.ArrayLinear = false;
}

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // OnEnable if first call
    if (_firstSetup) {
        OnEnable(materialEditor, properties);
        _firstSetup = false;
    }

    // Draw textures
}
```

DECLARATION

`public TextureArrayGUIDrawer(Material material, string arrayPropertyName, string assignedTexturesPropertyName, int textureCount, string fileName = null)`

PARAMETERS

Parameter	Description
material	The targeted material
arrayPropertyName	The reference name for the Array (Texture2DArray) shader property
assignedTexturesPropertyName	The reference name for the Assigned Textures (Float) shader property
textureCount	The max amount of textures this array will hold
fileName	Default: **Null** The filename of the Texture2DArray asset stored in a folder accompanying the material Used only on creation of the asset. Array is retrieved from the material afterwards

DESCRIPTION

Create a TextureArrayGUIDrawer using a material and property names

The Texture2DArray asset will be stored in a folder accompanying the material. Can be moved after creation

```
TextureArrayGUIDrawer _arrayDrawer;

private void OnEnable()
{
    // Using the material and property names is a good alternative when you dont have access to the material properties

    // Replace with your material and property names
    Material material;
    string arrayPropertyName = "_Array";
    string assignedTexturesPropertyName = "_AssignedTextures";

    // Create an array drawer
    // Texture amount should be changed to the max amount of textures you want in your array
    // In this example we will use a max of 4
    _arrayDrawer = new TextureArrayGUIDrawer(material, arrayPropertyName, assignedTexturesPropertyName, 4);

    // Settings of the arrays can be modified by these variables
    // Most of these settings will only take effect when the array is created
    // Array is not created on initialization, only when needed so these can be modified afterwards
    _arrayDrawerSection1.TextureFormat = TextureFormat.DXT5;
    _arrayDrawerSection1.TransferMipmaps = true;
    _arrayDrawerSection1.ArrayLinear = false;
}
```

DrawTexture

DECLARATION

`public Texture2D DrawTexture(int index, GUIContent content)`

`public Texture2D DrawTexture(Rect rect, int index, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
index	Index of the texture being changed in the desired array layout Not the index of the current array layout or textures, think of it as a constant within the set texture count
content	The GUIContent for the field

RETURNS

The texture input into the inspector

DESCRIPTION

Draws a texture using the assigned array

In some situations this will request user input when changing textures with a popup

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Draws the texture at index 0 with the label "Texture 1"
    Texture2D input = _arrayDrawer.DrawTexture(0, new GUIContent("Texture 1"));

    // Use the texture if needed
}
```

DrawTextureWithInt

DECLARATION

`public (Texture2D, int) DrawTextureWithInt(int index, int intValue, GUIContent content)`

`public (Texture2D, int) DrawTextureWithInt(Rect rect, int index, int intValue, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
index	Index of the texture being changed in the desired array layout Not the index of the current array layout or textures, think of it as a constant within the set texture count
intValue	The input integer value
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Int Value

DESCRIPTION

Draws a texture using the assigned array along with a integer value

In some situations this will request user input when changing textures with a popup

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Replace with your integer value
    int intValue;

    // Draws the texture with an integer field at index 0 with the label "Texture 1"
    (Texture2D, int) input = _arrayDrawer.DrawTextureWithInt(0, intValue, new GUIContent("Texture 1"));

    // Use the texture and int values if needed
    // Texture: input.Item1
    // Int: input.Item2
}
```

DrawTextureWithIntSlider

DECLARATION

`public (Texture2D, int) DrawTextureWithIntSlider(int index, int sliderValue, int sliderMin, int sliderMax, GUIContent content)`

`public (Texture2D, int) DrawTextureWithIntSlider(Rect rect, int index, int sliderValue, int sliderMin, int sliderMax, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
index	Index of the texture being changed in the desired array layout Not the index of the current array layout or textures, think of it as a constant within the set texture count
sliderValue	The input slider value
sliderMin	The minimum value that the slider will allow
sliderMax	The maximum value that the slider will allow
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Slider Value

DESCRIPTION

Draws a texture using the assigned array along with a integer slider

In some situations this will request user input when changing textures with a popup

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Replace with your slider value
    int slilderValue;

    // Draws the texture with an integer slider from 0-10 at index 0 with the label "Texture 1"
    (Texture2D, int) input = _arrayDrawer.DrawTextureWithIntSlider(0, slilderValue, 0, 10, new GUIContent("Texture 1"));

    // Use the texture and int values if needed
    // Texture: input.Item1
    // Int: input.Item2
}
```

DrawTextureWithFloat

DECLARATION

`public (Texture2D, float) DrawTextureWithFloat(int index, float floatValue, GUIContent content)`

`public (Texture2D, float) DrawTextureWithFloat(Rect rect, int index, float floatValue, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
index	Index of the texture being changed in the desired array layout Not the index of the current array layout or textures, think of it as a constant within the set texture count
floatValue	The input float value
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Float Value

DESCRIPTION

Draws a texture using the assigned array along with a float value

In some situations this will request user input when changing textures with a popup

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Replace with your float value
    float floatValue;

    // Draws the texture with a float field at index 0 with the label "Texture 1"
    (Texture2D, float) input = _arrayDrawer.DrawTextureWithFloat(0, floatValue, new GUIContent("Texture 1")));

    // Use the texture and float values if needed
    // Texture: input.Item1
    // Float: input.Item2
}
```

DrawTextureWithSlider

DECLARATION

`public (Texture2D, float) DrawTextureWithSlider(int index, float sliderValue, float sliderMin, float sliderMax, GUIContent content)`

`public (Texture2D, float) DrawTextureWithSlider(Rect rect, int index, float sliderValue, float sliderMin, float sliderMax, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
index	Index of the texture being changed in the desired array layout Not the index of the current array layout or textures, think of it as a constant within the set texture count
sliderValue	The input slider value
sliderMin	The minimum value that the slider will allow
sliderMax	The maximum value that the slider will allow
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Slider Value

DESCRIPTION

Draws a texture using the assigned array along with a slider

In some situations this will request user input when changing textures with a popup

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Replace with your slider value
    float floatValue;

    // Draws the texture with a float slider from 0-1 at index 0 with the label "Texture 1"
    (Texture2D, float) input = _arrayDrawer.DrawTexture(0, floatValue, 0, 1, new GUIContent("Texture 1")));

    // Use the texture and float values if needed
    // Texture: input.Item1
    // Float: input.Item2
}
```

DrawTextureWithColor

DECLARATION

`public (Texture2D, Color) DrawTextureWithColor(int index, Color colorValue, bool hdr, GUIContent content)`

`public (Texture2D, Color) DrawTextureWithColor(Rect rect, int index, Color colorValue, bool hdr, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
index	Index of the texture being changed in the desired array layout Not the index of the current array layout or textures, think of it as a constant within the set texture count
colorValue	The input color value
hdr	If the color is HDR
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Color Value

DESCRIPTION

Draws a texture using the assigned array along with a color value

In some situations this will request user input when changing textures with a popup

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Replace with your color value
    Color colorValue;

    // Draws the texture with a non-hdr color field at index 0 with the label "Texture 1"
    (Texture2D, float) input = _arrayDrawer.DrawTexture(0, colorValue, false, new GUIContent("Texture 1")));

    // Use the texture and color values if needed
    // Texture: input.Item1
    // Float: input.Item2
}
```

DrawTextures

DECLARATION

`public Texture2D[] DrawTextures(GUIContent[] content = null)`

PARAMETERS

Parameter	Description
content	Default: **Null** The GUIContent for each texture field in order If unassigned each texture will be named "Texture 1", "Texture 2", ...

RETURNS

Array of textures input into the inspector

DESCRIPTION

Draws all the textures in the array using DrawTexture

In some situations this will request user input when changing textures with a popup

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

// GUIContents for each texture field
// Used for labels and tooltips for the fields
// Any fields as null will automatically be assigned to "Texture {index + 1}"
GUIContent[] textureContents = new GUIContent[] {
    new GUIContent("Texture 1", "This is texture 1"),
    new GUIContent("Texture 2", "This is texture 2"),
    new GUIContent("Texture 3", "This is texture 3")
};

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Draws all the textures in the Texture Array using the predefined GUIContent[]
    Texture2D[] input = _arrayDrawer.DrawTextures(textureContents);

    // Use the texture values if needed
}
```

TextureAssignedAt

DECLARATION

`public bool TextureAssignedAt(int index)`

PARAMETERS

Parameter	Description
index	The index of the texture being checked

RETURNS

If the texture at the given index is assigned in the inspector

DESCRIPTION

Returns if a texture is assigned at the given index in the inspector

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Replace with your float value
    float floatValue;

    (Texture2D, float) input;
    // Draws a texture with a slider if the texture is set in the inspector
    // If not draw the texture field alone
    if(_arrayDrawer.TextureAssignedAt(0))
        input = _arrayDrawer.DrawTextureWithSlider(0, floatValue, 0, 1, new GUIContent("Texture 1"));
    else
        input.Item1 = _arrayDrawer.DrawTexture(0, new GUIContent("Texture 1"));

    // Use the texture and float values if neededed if needed
}
```

DeleteArray

DECLARATION

`public void DeleteArray()`

DESCRIPTION

Clears the array and deletes its file and folder if empty

Does not prevent drawing further textures, acts more like a reset

```
// Assumes TextureArrayGUIDrawer object is already created
TextureArrayGUIDrawer _arrayDrawer;

public override void OnGUI(MaterialEditor materialEditor, MaterialProperty[] properties)
{
    // Button that clears the array
    // Will clear the array file and its folder if required
    // Good to use as a reset button for the textures, will not lock further texture drawing
    if(GUILayout.Button("Clear Texture Array")) {
        _arrayDrawer.DeleteArray();
    }
}
```

4.3.2 GUIUtilities.GUIUtilities

Description

`Unity Editor Only`

Many GUI functions to make creation easier without MaterialProperties including:

- Texture fields including fields accompanied by:
- Sliders
- Int/Float fields
- Color field
- Vector2 fields without the extra space
- Horizontal/Vertical Backgrounds
- Single line helpers

Variables

Variable	Description
LINE_HEIGHT	Value: **18** The height that one line takes in the inspector
LINE_SPACING	Value: **2** The height in-between two lines in the inspector

GetLineRect

DECLARATION

`public static Rect GetLineRect(float heightOverride = LINE_HEIGHT + LINE_SPACING)`

PARAMETERS

Parameter	Description
heightOverride	Default: **20 (LINE_HEIGHT + LINE_SPACING)** Overrides the height of the output rect

RETURNS

The rect of a single line

DESCRIPTION

Gets the rect of a single line, updating the GUILayout accordingly

```
private void OnGUI()
{
    // Gets the rect for the current line updating the GUILayout accordingly
    Rect lineRect = GUIUtilities.GetLineRect();
}
```

DrawTexture2DArray

DECLARATION

`public static Texture2DArray DrawTexture2DArray(Texture2DArray array, GUIContent content)` `public static Texture2DArray DrawTexture2DArray(Rect rect, Texture2DArray array, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
array	The input Texture2DArray
content	The GUIContent for the field

RETURNS

The Texture2DArray input into the inspector

DESCRIPTION

Draws a Texture2DArray field

```
// Replace with your array value
Texture2DArray _array;

private void OnGUI()
{
    // Draws a Texture2DArray field with the label "Texture Array"
    _array = GUIUtilities.DrawTexture2DArray(_array, new GUIContent("Texture Array"));
}
```

DrawTexture

DECLARATION

`public static Texture2D DrawTexture(Texture2D texture, GUIContent content)`

`public static Texture2D DrawTexture(Rect rect, Texture2D texture, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
texture	The input texture
content	The GUIContent for the field

RETURNS

The texture input into the inspector

DESCRIPTION

Draws a single texture field

```
// Replace with your texture value
Texture2D _texture;

private void OnGUI()
{
    // Draws a Texture2D field with the label "Texture"
    _texture = GUIUtilities.DrawTexture(_texture, new GUIContent("Texture"));
}
```

DrawTextureWithInt

DECLARATION

`public static (Texture2D, int) DrawTextureWithInt(Texture2D texture, int intValue, GUIContent content)`

`public static (Texture2D, int) DrawTextureWithInt(Rect rect, Texture2D texture, int intValue, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
texture	The input texture
intValue	The input integer value
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Int Value

DESCRIPTION

Draws a single texture field with an accompanied integer value

```
// Replace with your texture/integer values
Texture2D _texture;
int _integer;

private void OnGUI()
{
    // Draws a Texture2D field with an integer value with the label "Texture"
    (Texture2D, int) input = GUIUtilities.DrawTextureWithInt(_texture, _integer, new GUIContent("Texture"));

    // Retrieve the values from the input
    _texture = input.Item1;
    _integer = input.Item2;
}
```

DrawTextureWithIntSlider

DECLARATION

`public static (Texture2D, int) DrawTextureWithIntSlider(Texture2D texture, int sliderValue, int sliderMin, int sliderMax, GUIContent content)`

`public static (Texture2D, int) DrawTextureWithIntSlider(Rect rect, Texture2D texture, int sliderValue, int sliderMin, int sliderMax, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
texture	The input texture
sliderValue	The input slider value
sliderMin	The minimum value that the slider will allow
sliderMax	The maximum value that the slider will allow
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Slider Value

DESCRIPTION

Draws a single texture field with an accompanied integer slider

```
// Replace with your texture/integer values
Texture2D _texture;
int _integer;

private void OnGUI()
{
    // Draws a Texture2D field with an integer slider with the label "Texture"
    // This example has the slider from 0-10
    (Texture2D, int) input = GUIUtilities.DrawTextureWithIntSlider(_texture, _integer, 0, 10, new GUIContent("Texture"));

    // Retrieve the values from the input
    _texture = input.Item1;
    _integer = input.Item2;
}
```

DrawTextureWithFloat

DECLARATION

`public static (Texture2D, float) DrawTextureWithFloat(Texture2D texture, float floatValue, GUIContent content)`

`public static (Texture2D, float) DrawTextureWithFloat(Rect rect, Texture2D texture, float floatValue, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
texture	The input texture
floatValue	The input float value
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Float Value

DESCRIPTION

Draws a single texture field with an accompanied float value

```
// Replace with your texture/float values
Texture2D _texture;
float _float;

private void OnGUI()
{
    // Draws a Texture2D field with an float value with the label "Texture"
    (Texture2D, float) input = GUIUtilities.DrawTextureWithFloat(_texture, _float, new GUIContent("Texture"));

    // Retrieve the values from the input
    _texture = input.Item1;
    _float = input.Item2;
}
```

DrawTextureWithSlider

DECLARATION

`public static (Texture2D, float) DrawTextureWithSlider(Texture2D texture, float sliderValue, float sliderMin, float sliderMax, GUIContent content)` `public static (Texture2D, float) DrawTextureWithSlider(Rect rect, Texture2D texture, float sliderValue, float sliderMin, float sliderMax, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
texture	The input texture
sliderValue	The input slider value
sliderMin	The minimum value that the slider will allow
sliderMax	The maximum value that the slider will allow
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Slider Value

DESCRIPTION

Draws a single texture field with an accompanied slider

```
// Replace with your texture/float values
Texture2D _texture;
float _float;

private void OnGUI()
{
    // Draws a Texture2D field with an float slider with the label "Texture"
    // This example has the slider from 0-1
    (Texture2D, int) input = GUIUtilities.DrawTextureWithSlider(_texture, _integer, 0, 1, new GUIContent("Texture"));

    // Retrieve the values from the input
    _texture = input.Item1;
    _float = input.Item2;
}
```

DrawTextureWithColor

DECLARATION

`public static (Texture2D, Color) DrawTextureWithColor(Texture2D texture, Color color, bool hdr, GUIContent content)`

`public static (Texture2D, Color) DrawTextureWithColor(Rect rect, Texture2D texture, Color color, bool hdr, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
texture	The input texture
colorValue	The input color value
hdr	If the color is HDR
content	The GUIContent for the field

RETURNS

Item1: Texture, Item2: Color Value

DESCRIPTION

Draws a single texture field with an accompanied color value

```
// Replace with your texture/color values
Texture2D _texture;
Color _color;

private void OnGUI()
{
    // Draws a Texture2D field with an color value with the label "Texture"
    // This example has the color picker non-hdr
    (Texture2D, Color) input = GUIUtilities.DrawTextureWithColor(_texture, _color, false, new GUIContent("Texture"));

    // Retrieve the values from the input
    _texture = input.Item1;
    _color = input.Item2;
}
```

DrawTextures

DECLARATION

`public static Texture2D[] DrawTextures(Texture2D[] textures, GUIContent[] content = null)`

PARAMETERS

Parameter	Description
textures	The input textures that will be drawn in order
content	Default: **null** The GUIContent for each field in order If unassigned each texture will be named "Texture 1", "Texture 2", ...

RETURNS

Array of textures input into the inspector

DESCRIPTION

Draws multiple texture fields in sequence from the given texture array

```
// Replace with your texture values
Texture2D[] _textures;

// Example of assigning the GUIContent
GUIContent[] _texturesContent;

private void OnEnable()
{
    // Initialize the GUIContent array
    // In this example there are 4 textures, so create a GUIContent for each
    // Any fields as null will automatically be assigned to "Texture {index + 1}"
    _texturesContent = {
        new GUIContent("Texture 1"),
        new GUIContent("Texture 2"),
        new GUIContent("Texture 3"),
        new GUIContent("Texture 4")
    };
}

private void OnGUI()
{
    // Draws a Texture2D field with the label "Texture"
    // In this example the GUIContent is setup in OnEnable
    // If the GUIContent is set to null each texture will be labelled "Texture 1", "Texture 2", ...
    _textures = GUIUtilities.DrawTextures(_texture, _texturesContent);
}
```

DrawVector2Field

DECLARATION

`public static Vector2 DrawVector2Field(Vector2 value, GUIContent content)`

`public static Vector2 DrawVector2Field(Rect rect, Vector2 value, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
value	The input Vector2 value
content	The GUIContent for the field

RETURNS

The Vector2 input into the inspector

DESCRIPTION

Draws a Vector2 field removing the space at the end which EditorGUI.Vector2Field draws

```
// Replace with your Vector2 value
Vector2 _vector2;

private void OnGUI()
{
    // Draws a Vector2 field with the label "Vector 2"
    _vector2 = GUIUtilities.DrawVector2Field(_vector2, new GUIContent("Vector 2"));
}
```

DrawVector2IntField

DECLARATION

`public static Vector2Int DrawVector2IntField(Vector2Int value, GUIContent content)`

`public static Vector2Int DrawVector2IntField(Rect rect, Vector2Int value, GUIContent content)`

PARAMETERS

Parameter	Description
rect	The space that the field will use
value	The input Vector2Int value
content	The GUIContent for the field

RETURNS

The Vector2Int input into the inspector

DESCRIPTION

Draws a Vector2Int field removing the space at the end which EditorGUI.Vector2IntField draws

```
// Replace with your Vector2 value
Vector2Int _vector2Int;

private void OnGUI()
{
    // Draws a Vector2 field with the label "Vector 2 Int"
    _vector2Int = GUIUtilities.DrawVector2IntField(_vector2Int, new GUIContent("Vector 2 Int"));
}
```

BeginBackgroundVertical

DECLARATION

`public static void BeginBackgroundVertical()`

DESCRIPTION

Begins a background in the same style as a HelpBox using GUILayout.BeginVertical

```
private void OnGUI()
{
    // Begin the vertical background
    GUIUtilities.BeginBackgroundVertical();

    // Execute GUI that is inside the background

    // End the vertical background
    GUIUtilities.EndBackgroundVertical();
}
```

EndBackgroundVertical

DECLARATION

`public static void EndBackgroundVertical()`

DESCRIPTION

Ends the vertical background, same as calling GUILayout.EndVertical

```
private void OnGUI()
{
    // Begin the vertical background
    GUIUtilities.BeginBackgroundVertical();

    // Execute GUI that is inside the background

    // End the vertical background
    GUIUtilities.EndBackgroundVertical();
}
```

BeginBackgroundHorizontal

DECLARATION

`public static void BeginBackgroundHorizontal()`

DESCRIPTION

Begins a background in the same style as a HelpBox using GUILayout.BeginHorizontal

```
private void OnGUI()
{
    // Begin the horizontal background
    GUIUtilities.BeginBackgroundHorizontal();

    // Execute GUI that is inside the background

    // End the horizontal background
    GUIUtilities.EndBackgroundHorizontal();
}
```

EndBackgroundHorizontal

DECLARATION

`public static void EndBackgroundHorizontal()`

DESCRIPTION

Ends the horizontal background, same as calling GUILayout.EndHorizontal

```
private void OnGUI()
{
    // Begin the horizontal background
    GUIUtilities.BeginBackgroundHorizontal();

    // Execute GUI that is inside the background

    // End the horizontal background
    GUIUtilities.EndBackgroundHorizontal();
}
```

4.4 CustomDialog

4.4.1 CustomDialog.ShaderGUIDialog

Description

`Unity Editor Only`

Wrapper to remove the "PropertiesGUI() is being called recursively" warning when calling EditorUtility.DisplayDialog in OnGUI within a ShaderGUI inherited class

Example of using the ShaderGUIDialog

```
private void OnGUI()
{
    // Get input
    int input = ShaderGUIDialog.DisplayDialogComplex(
        "Title",
        "Message",
        "Ok",
        "Cancel",
        "Alt"
    )

    // Use input
    switch(input) {
        case 0:
            Debug.Log("OK was selected");
            break;
        case 1:
            Debug.Log("CANCEL was selected");
            break;
        case 2:
            Debug.Log("ALT was selected");
            break;
    }
}
```

DisplayDialog

DECLARATION

`public static bool DisplayDialog(string title, string message, string ok, string cancel)`

PARAMETERS

Parameter	Description
title	Title of the window
message	Message displayed in the dialog
ok	Left button underneath the message
cancel	Right button underneath the message

RETURNS

If the ok button was pressed

DESCRIPTION

Displays a modal dialog removing the ShaderGUI specific warning

DisplayDialogComplex

DECLARATION

```
public static int DisplayDialogComplex(string title, string message, string ok, string cancel, string alt)
```

PARAMETERS

Parameter	Description
title	Title of the window
message	Message displayed in the dialog
ok	Left button underneath the message
cancel	Right button underneath the message
alt	Middle button underneath the message

RETURNS

0, 1, 2 for ok, cancel, alt responses respectively

DESCRIPTION

Displays a modal dialog with three buttons removing the ShaderGUI specific warning

4.5 CustomWindows

4.5.1 CustomWindows.Texture2DArrayWindowBase

Description

`Unity Editor Only`

Base class for creating the Create/Configure Array windows

Variables

Variable	Description
_textures	The textures that are assigned in the window Should be updated in tandem with _texturesResizing & _textureErrors
_texturesResizing	**AUTOMATICALLY UPDATES** Which textures have been marked as resizing Should be updated in tandem with *textures* & _textureErrors
_textureErrors	**AUTOMATICALLY UPDATES** Which textures have an error that needs to be addressed Should be updated in tandem with *textures* & _texturesResizing
_arrayTextureFormatIndex	Default: **Index Of TextureFormat.DXT5** The format index of the output array *Modifiable in the window*
_arrayMipMaps	Default: **true** If the output array will transfer mipmaps from the inputted textures *Modifiable in the window*
_arrayLinear	Default: **false** If the output array will be linear Recommended in the Built-In Render Pipeline only when including normal maps **Not Recommended in URP/HDRP as it will result in brighter textures** *Modifiable in the window*
_arrayWrapMode	Default: **TextureWrapMode.Repeat** The Wrap Mode of the output array *Modifiable in the window*
_arrayFilterMode	Default: **FilterMode.Bilinear** The Filter Mode of the output array *Modifiable in the window*
_arrayAnisoLevel	Default: **1** The Aniso Level of the output array *Modifiable in the window*
_resizeTextures	Default: **False** If all the inputted textures will be resized to _arrayResolution *Modifiable in the window*
_arrayResolution	Default: **First Assigned Texture Resolution** The resolution that all inputted textures will be resized to if _resizeTextures is enabled *Displayed and modifiable in the window when _resizeTextures enabled*
_resizeFilterMode	Default: **FilterMode.Bilinear** The filter mode used when resizing textures *Displayed and modifiable in the window when _resizeTextures enabled or _texturesResizing has any values set to true*
_textureFormatSettingEnabled	Default: **True** If the Texture Format setting appears in the window
_mipmapsSettingEnabled	Default: **True** If the Generate Mipmaps setting appears in the window
_linearSettingEnabled	Default: **True** If the Linear setting appears in the window
_wrapModeSettingEnabled	Default: **True** If the Wrap Mode setting appears in the window
_filterModeSettingEnabled	

Variable	Description
	Default: **True** If the Filter Mode setting appears in the window
_anisoLevelSettingEnabled	Default: **True** If the Aniso Level setting appears in the window
_resizeSettingEnabled	Default: **True** If the Resize Textures setting appears in the window
_headerStyle	Style used for headers in the GUI

Example

Custom window example with a create array button

```
using UnityEngine;
using TextureArrayUtilities.CustomWindows;

#if UNITY_EDITOR
using UnityEditor;

public class CustomArrayWindow : Texture2DArrayWindowBase
{
    // Creates the window
    protected static void ShowWindow()
    {
        CustomArrayWindow window = GetWindow<CustomArrayWindow>(false, "Custom Window");
        window.Show();
    }

    // Setting the MenuItem as "Assets/..." will allow opening the window through the right click menu in the Project window
    [MenuItem("Assets/CustomArrayWindow")]
    static void CreateWindowInProjectContextMenu(MenuCommand command)
    {
        ShowWindow();
    }

    // Setting the MenuItem as "Tools/..." for example will allow opening the window through the toolbar
    [MenuItem("Tools/CustomArrayWindow")]
    static void CreateWindowInToolbar(MenuCommand command)
    {
        ShowWindow();
    }

    protected override void OnGUIUpdate()
    {
        // All the array textures and settings are drawn in base.OnGUIUpdate()
        // Everything can be modified through the
        base.OnGUIUpdate();

        // Example button to create the array in the Assets path
        if(GUILayout.Button("Create Array")) {
            CreateArray("Assets/TextureArray.asset");

            // Close Window
            Close();
        }
    }
}

#endif
```

CreateGUI

DECLARATION

`protected virtual void CreateGUI()`

DESCRIPTION

Called when the GUI is first created

base.CreateGUI(); Must be called before performing operations

```
protected override void CreateGUI()
{
    base.CreateGUI();

    // Code executed when GUI first created
}
```

OnGUIUpdate

DECLARATION

`protected virtual void OnGUIUpdate()`

DESCRIPTION

Called every GUI update, used due to other tasks in OnGUI

base.OnGUI(); Will draw the textures, settings, and output sections

```
protected override void OnGUIUpdate()
{
    // All code executed here will be called for each update in OnGUI

    // GUI at the top of the window performed before base

    base.OnGUIUpdate();

    // GUI at the bottom of the window performed after base
}
```

CalculateElementHeight

DECLARATION

`protected virtual float CalculateElementHeight(int index)`

PARAMETERS

Parameter	Description
index	Index of the texture list which the height is being calculated for the ReorderableList GUI

RETURNS

Height that the element will use

DESCRIPTION

Calculates the height of an element for the ReorderableList GUI where the textures are assigned

base.CalculateElementHeight(); Must be called before performing operations and will return the pre-calculated height

```
protected override float CalculateElementHeight(int index)
{
    float height = base.CalculateElementHeight(index);

    // Calculate element height here

    return height;
}
```

DrawTextureField

DECLARATION

`protected virtual void DrawTextureField(Rect rect, int index, bool isActive, bool isFocused)`

PARAMETERS

Parameter	Description
rect	Rect with all the available space in the element Available space is calculated and can be changed in CalculateElementHeight
index	The index of the current element in the textures list
isActive	If this element is is being moved or interacted with in the GUI
isFocused	If this element is selected in the GUI

DESCRIPTION

Draws a texture element in the textures list

When modifying the height in any way Repaint(); should be called to update it on the same GUI call, otherwise will be updated on the next

```
protected override float CalculateElementHeight(int index)
{
    float height = base.CalculateElementHeight(index);

    // Calculate element height here

    return height;
}
```

DrawArraySettings

DECLARATION

`protected virtual void DrawArraySettings()`

DESCRIPTION

Draws the settings of the output array. Settings that are enabled here can be toggles with the respective settingEnabled variables

All GUI called here will be drawn within the Array Settings section

base.DrawArraySettings() will draw the settings

```
protected override void DrawArraySettings()
{
    // GUI before the main settings within the Array Settings Section

    base.DrawArraySettings();

    // GUI after the main settings within the Array Settings Section
}
```

DrawFinalOutputFields

DECLARATION

`protected virtual void DrawFinalOutputFields()`

DESCRIPTION

Draws the details of the output array

All GUI called here will be drawn within the Final Output section

base.DrawFinalOutputFields() will draw the final output details

```
protected override void DrawFinalOutputFields()
{
    // GUI before the main output details the Final Output Section
    base.DrawFinalOutputFields();

    // GUI after the main output details within the Final Output Section
}
```

DisplayWarnings

DECLARATION

`protected virtual void DisplayWarnings()`

DESCRIPTION

Displays any warnings for array creation

All GUI called here will be drawn within and at the bottom of the Final Output section

base.DisplayWarnings() must be called before showing any extra warnings

```
protected override void DisplayWarnings()
{
    base.DisplayWarnings();

    // Display any extra warnings here

    // Example
    // if(warningCondition) {
    //     EditorGUILayout.HelpBox("Warning", MessageType.Warning);
    // }
}
```

OnTextureAdd

DECLARATION

`protected virtual void OnTextureAdd(ReorderableList list)`

PARAMETERS

Parameter	Description
list The ReorderableList used for displaying the GUI s	

DESCRIPTION

Called when the add button is clicked in the ReorderableList GUI

base.OnTextureAdd() will add a texture to the texture lists and should be called

```
protected override void OnTextureAdd(ReorderableList list)
{
    // Code executed before a texture is added

    base.OnTextureAdd(list);

    // Code executed after a texture is added
}
```

OnTextureRemove

DECLARATION

`protected virtual void OnTextureRemove(ReorderableList list)`

PARAMETERS

Parameter	Description
list	The ReorderableList used for displaying the GUI

DESCRIPTION

Called when the remove button or delete key is pressed in the ReorderableList GUI

base.OnTextureRemove() will remove the selected texture from the texture lists and should be called

```
protected override void OnTextureRemove(ReorderableList list)
{
    // Code executed before a texture is removed

    base.OnTextureRemove(list);

    // Code executed after a texture is removed
}
```

OnTexturesReorder

DECLARATION

`protected virtual void OnTexturesReorder(ReorderableList list, int oldActiveElement, int newActiveElement)`

PARAMETERS

Parameter	Description
list	The ReorderableList used for displaying the GUI
oldActiveElement	The previous index of the reordered element
newActiveElement	The new index of the reordered element

DESCRIPTION

Called when the textures are reordered in the ReorderableList GUI

base.OnTexturesReorder() will handle reordering the texture lists and should be called

```
protected override void OnTexturesReorder(ReorderableList list, int oldActiveElement, int newActiveElement)
{
    // Code executed before a texture is removed

    base.OnTexturesReorder(list, oldActiveElement, newActiveElement);

    // Code executed after a texture is removed
}
```

OnDragUpdate

DECLARATION

`protected virtual void OnDragUpdate(Object[] objectReferences)`

PARAMETERS

Parameter	Description
objectReferences	The items that are being dragged

DESCRIPTION

Called every update while anything is dragged over the window

base.OnDragUpdate(); Must be called before performing operations

```
protected override void OnDragUpdate(Object[] objectReferences)
{
    base.OnDragUpdate(objectReferences);

    // Handle dragged items
}
```

OnDragPerform

DECLARATION

`protected virtual void OnDragPerform(Object[] objectReferences)`

PARAMETERS

Parameter	Description
objectReferences	The items that are being dragged

DESCRIPTION

Called when the drag is complete over the window

base.OnDragPerform(); Must be called before performing operations

```
protected override void OnDragPerform(Object[] objectReferences)
{
    base.OnDragPerform(objectReferences);

    // Handle dragged items
}
```

ArrayTexturesExist

DECLARATION

`protected bool ArrayTexturesExist()`

RETURNS

If any textures exist in the list

DESCRIPTION

Checks if any textures exist in the list

CreateArray

DECLARATION

`protected void CreateArray(string path)`

RETURNS

If the operation was successful

PARAMETERS

Parameter	Description
path	The file location that the array will be created within the project folder Should always start with "Assets/"

DESCRIPTION

Creates the array using the specified textures and settings at a given path

Example creating an array with a button

```
protected override void OnGUIUpdate()
{
    // Draw GUI
    base.OnGUIUpdate();

    // Example button to create the array in the Assets path
    if(GUILayout.Button("Create Array")) {
        CreateArray("Assets/TextureArray.asset");

        // Close Window
        Close();
    }
}
```

4.6 Compression

4.6.1 Compression.BooleanCompression

Description

Used to compress an array of boolean values into a single integer

Example of using BooleanCompression

```
private void Function()
{
    // Replace with your boolean array of values
    // These values are used for this example
    bool[] valuesToCompress = {
        true,
        false,
        false,
        true
    };

    // This will return the values in a compressed form in an integer
    // In this example compression returns "9" using this specific layout of values
    // Values can be retrieved using BooleanCompression.GetValues/GetValue
    int compressedValues = BooleanCompression.CompressValues(valuesToCompress);

    // Example of retrieving the values into an array
    // Using the previous array length as the total count
    bool[] retrievedValues = BooleanCompression.GetValues(compressedValues, valuesToCompress.Length);

    // Example of retrieving the values individually
    bool value1 = BooleanCompression.GetValue(compressedValues, 0);
    bool value2 = BooleanCompression.GetValue(compressedValues, 1);
    bool value3 = BooleanCompression.GetValue(compressedValues, 2);
    bool value4 = BooleanCompression.GetValue(compressedValues, 3);
}
```

CompressValues

DECLARATION

`public static int CompressValues(params bool[] values)`

PARAMETERS

Parameter	Description
values	Bools that will be compressed

RETURNS

Compressed int of bools

DESCRIPTION

Compresses the input array of bools into an int. Values can be retrieved using GetValues and GetValue

```
private void Function()
{
    // Replace with your boolean array of values
    // These values are used for this example
    bool[] valuesToCompress = {
        true,
        false,
        false,
        true
    };

    // This will return the values in a compressed form in an integer
    // In this example compression returns "9" using this specific layout of values
    // Values can be retrieved using GetValues/GetValue
    int compressedValues = BooleanCompression.CompressValues(valuesToCompress);
}
```

GetValues

DECLARATION

`public static bool[] GetValues(int compressedValues, int valueCount)`

PARAMETERS

Parameter	Description
compressedValues	Compressed int of bool values
valueCount	The total amount of values stored in the compressedValues

RETURNS

Array of all the values stored in the input comrpressedValues

DESCRIPTION

Retrieves all the boolean values stored in a compressed int created through CompressValues

```
private void Function()
{
    // Replace with your compressed values
    // Compressed values can be retrieved through CompressValues
    int compressedValues;

    // Retrieves the values from the integer into an array
    // Using 4 for this example but the valueCount should be the amount of values stored in the compressedValues
    bool[] retrievedValues = BooleanCompression.GetValues(compressedValues, 4);

    // If you have the original array you can use its length for the valueCount
    // bool[] retrievedValues = BooleanCompression.GetValues(compressedValues, valuesArray.Length);
}
```

AddValue

DECLARATION

`public static int AddValue(int compressedValues, int index, bool value)`

PARAMETERS

Parameter	Description
compressedValues	Compressed int of bool values
index	Index that the input value will overwrite
value	Value inserted into the compressedValue

RETURNS

New compressed integer with the added value

DESCRIPTION

Overwrites a value at a given index into the inputted compressedValues created through CompressValues

```
private void Function()
{
    // Replace with your compressed values
    // Compressed values can be retrieved through CompressValues
    int compressedValues;

    // Add a value at a specific index into the compressedValues
    // In this example add the value "true" to index 5
    // This will overwrite the previous value at this index
    // The index can also be larger than the current value amount in the compressedValues, it will be expanded accordingly
    compressedValues = BooleanCompression.AddValue(compressedValues, 5, true);
}
```

GetValue

DECLARATION

```
public static bool GetValue(int compressedValues, int index)
```

PARAMETERS

Parameter	Description
compressedValues	Compressed int of bool values
index	Index of which bool value will be returned

RETURNS

Value at the given index

DESCRIPTION

Gets a compressed value at a given index from the inputted compressedValues created through CompressValues

```
private void Function()
{
    // Replace with your compressed values
    // Compressed values can be retrieved through CompressValues
    int compressedValues;

    // Retrieve a value at a specific index from the compressedValues
    // In this example retrieve the value at index 2
    bool value3 = BooleanCompression.GetValue(compressedValues, 2);
}
```