```
        .gitignore
        /[Ll]ibrary/
/[Tt]emp/
/[Oo]bj/
/[Bb]uild/
/[Bb]uilds/
/[Ll]ogs/
/[Uu]ser[Ss]ettings/
/[Mm]emoryCaptures/
/[Rr]ecordings/
/[Aa]ssets/Plugins/Editor/JetBrains*
.vs/
.gradle/
ExportedObj/
.consulo/
sysinfo.txt
crashlytics-build.properties
/[Aa]ssets/[Aa]ddressable[Aa]ssets[Dd]ata/*/*.bin*
/[Aa]ssets/[Ss]treamingAssets/aa.meta
/[Aa]ssets/[Ss]treamingAssets/aa/*
        AmplifyShaderEditor.asmdef.meta
        fileFormatVersion: 2
guid: f540dafdfbc0586439d98823585550d4
AssemblyDefinitionImporter:
    externalObjects: {}
    userData:
    assetBundleName:
    assetBundleVariant:
        ChangeLog.txt.meta
        fileFormatVersion: 2
guid: 580cccd3e608b7f4cac35ea46d62d429
timeCreated: 1481127071
licenseType: Store
TextScriptImporter:
    userData:
    assetBundleName:
    assetBundleVariant:
        CreatingTerrainsWithASE.txt.meta
        fileFormatVersion: 2
guid: f11d5aaf59fc38544b841924280 1ff97
timeCreated: 1513615640
licenseType: Store
TextScriptImporter:
    userData:
    assetBundleName:
    assetBundleVariant:
        Credits.txt.meta
        fileFormatVersion: 2
guid: 451790f45b4e5434586d16e924540ee7
timeCreated: 1481127071
licenseType: Store
TextScriptImporter:
    userData:
    assetBundleName:
    assetBundleVariant:
        DownloadSamplesInstructions.txt.meta
        fileFormatVersion: 2
guid: 03fe3bdcb7262a84ca060ff336c7d8d1
timeCreated: 1481127071
licenseType: Store
TextScriptImporter:
    userData:
    assetBundleName:
    assetBundleVariant:
        README Samples.txt.meta
        fileFormatVersion: 2
guid: 5c5208e121a880c4fbb9de7aaeaa8aa2
timeCreated: 1541776170
licenseType: Store
TextScriptImporter:
    userData:
    assetBundleName:
    assetBundleVariant:
        Examples.meta
        fileFormatVersion: 2
guid: 425c3aed47dd05444960ca41af18e591
folderAsset: yes
timeCreated: 1481126943
licenseType: Store
DefaultImporter:
    userData:
    assetBundleName:
    assetBundleVariant:
        ActionData.cs
        using System;
using UnityEngine;
namespace AmplifyShaderEditor
{
        public class ActionData
        {
                public virtual void ExecuteForward() { }
                public virtual void ExecuteReverse() { }
        }
```

```
public class CreateNodeActionData : ActionData
{
	private int m_nodeId;
	private System.Type m_nodeType;
	private Vector2 m_nodePos;
	public CreateNodeActionData( ParentNode node )
	{
		m_nodeId = node.UniqueId;
		m_nodePos = node.Vec2Position;
		m_nodeType = node.GetType();
	}
	public CreateNodeActionData( int nodeId, System.Type nodeType, Vector2 nodePos )
	{
		m_nodeId = nodeId;
		m_nodePos = nodePos;
		m_nodeType = nodeType;
	}
	public override void ExecuteForward()
	{
		UIUtils.CreateNode( m_nodeType, false, m_nodePos, m_nodeId );
	}
	public override void ExecuteReverse()
	{
		UIUtils.DestroyNode( m_nodeId );
	}
	public override string ToString()
	{
		return "Create Node - Type: " + m_nodeType + " Node: " + m_nodeId + " Position: " + m_nodePos;
	}
}
public class DestroyNodeActionData : ActionData
{
	private int m_nodeId;
	private System.Type m_nodeType;
	private Vector2 m_nodePos;
	public DestroyNodeActionData( ParentNode node )
	{
		m_nodeId = node.UniqueId;
		m_nodePos = node.Vec2Position;
		m_nodeType = node.GetType();
	}
	public DestroyNodeActionData( int nodeId, System.Type nodeType, Vector2 nodePos )
	{
		m_nodeId = nodeId;
		m_nodePos = nodePos;
		m_nodeType = nodeType;
	}
	public override void ExecuteForward()
	{
		UIUtils.DestroyNode( m_nodeId );
	}
	public override void ExecuteReverse()
	{
		UIUtils.CreateNode( m_nodeType, false, m_nodePos, m_nodeId );
	}
	public override string ToString()
	{
		return "Destroy Node - Type: " + m_nodeType + " Node: " + m_nodeId + " Position: " + m_nodePos;
	}
}
public class MoveNodeActionData : ActionData
{
	private int m_nodeId;
	private Vector2 m_nodeInitalPos;
	private Vector2 m_nodeFinalPos;
	public MoveNodeActionData( int nodeId, Vector2 nodeInitialPos, Vector2 nodeFinalPos )
	{
		m_nodeId = nodeId;
		m_nodeInitalPos = nodeInitialPos;
		m_nodeFinalPos = nodeFinalPos;
	}
	public override void ExecuteForward()
	{
		ParentNode node = UIUtils.GetNode( m_nodeId );
		if ( node != null )
			node.Vec2Position = m_nodeFinalPos;
	}
	public override void ExecuteReverse()
	{
		ParentNode node = UIUtils.GetNode( m_nodeId );
		if ( node != null )
			node.Vec2Position = m_nodeInitalPos;
	}
	public override string ToString()
	{
		return "Move Node - Node: " + m_nodeId + " Initial Position: " + m_nodeInitalPos + " Final Position: " + m_nodeFinalPos;
	}
}
public class CreateConnectionActionData : ActionData
{
	private int m_inputNodeId;
	private int m_inputPortId;
	private int m_outputNodeId;
```

```
		private int m_outputPortId;
		public CreateConnectionActionData( int inputNodeId, int inputPortId, int outputNodeId, int outputPortId )
		{
			m_inputNodeId = inputNodeId;
			m_inputPortId = inputPortId;
			m_outputNodeId = outputNodeId;
			m_outputPortId = outputPortId;
		}
		public override void ExecuteForward()
		{
			UIUtils.ConnectInputToOutput( m_inputNodeId, m_inputPortId, m_outputNodeId, m_outputPortId );
		}
		public override void ExecuteReverse()
		{
			UIUtils.DeleteConnection( true, m_inputNodeId, m_inputPortId, false, true );
		}
		public override string ToString()
		{
			return "Create Connection Node - Input Node: " + m_inputNodeId + " Input Port: " + m_inputPortId + " Output Node: " + m_outputNodeId + " Output Port: " + m_outputPortId;
		}
}
public class DestroyConnectionActionData : ActionData
{
		private int m_inputNodeId;
		private int m_inputPortId;
		private int m_outputNodeId;
		private int m_outputPortId;
		public DestroyConnectionActionData( int inputNodeId, int inputPortId, int outputNodeId, int outputPortId )
		{
			m_inputNodeId = inputNodeId;
			m_inputPortId = inputPortId;
			m_outputNodeId = outputNodeId;
			m_outputPortId = outputPortId;
		}
		public override void ExecuteForward()
		{
			UIUtils.DeleteConnection( true, m_inputNodeId, m_inputPortId, false, true );
		}
		public override void ExecuteReverse()
		{
			UIUtils.ConnectInputToOutput( m_inputNodeId, m_inputPortId, m_outputNodeId, m_outputPortId );
		}
		public override string ToString()
		{
			return "Destroy Connection Node - Input Node: " + m_inputNodeId + " Input Port: " + m_inputPortId + " Output Node: " + m_outputNodeId + " Output Port: " + m_outputPortId;
		}
}
public class MoveInputConnectionActionData : ActionData
{
		private int m_oldInputNodeId;
		private int m_oldInputNodePortId;
		private int m_newInputNodeId;
		private int m_newInputNodePortId;
		private int m_outputNodeId;
		private int m_outputPortId;
		public MoveInputConnectionActionData( int oldInputNodeId, int oldInputPortId, int newInputNodeId, int newInputPortId, int outputNodeId, int outputPortId )
		{
			m_oldInputNodeId = oldInputNodeId;
			m_oldInputNodePortId = oldInputPortId;
			m_newInputNodeId = newInputNodeId;
			m_newInputNodePortId = newInputPortId;
			m_outputNodeId = outputNodeId;
			m_outputPortId = outputPortId;
		}
		public override void ExecuteForward()
		{
			UIUtils.DeleteConnection( true, m_oldInputNodeId, m_oldInputNodePortId, false, true );
			UIUtils.ConnectInputToOutput( m_newInputNodeId, m_newInputNodePortId, m_outputNodeId, m_outputPortId );
		}
		public override void ExecuteReverse()
		{
			base.ExecuteReverse();
			UIUtils.DeleteConnection( true, m_newInputNodeId, m_newInputNodePortId, false, true );
			UIUtils.ConnectInputToOutput( m_oldInputNodeId, m_oldInputNodePortId, m_outputNodeId, m_outputPortId );
		}
		public override string ToString()
		{
			return "Move Input Connection Node - Old Input Node: " + m_oldInputNodeId + " Old Input Port: " + m_oldInputNodePortId + " New Input Node: " + m_newInputNodeId + " New Input Port: " + m_newInputNodePortId + " Output Node: " + m_outputNodeId + " Output Port: " + m_outputPortId;
		}
}
public class MoveOutputConnectionActionData : ActionData
{
		private int m_inputNodeId;
		private int m_inputPortId;
		private int m_newOutputNodeId;
		private int m_newOutputPortId;
		private int m_oldOutputNodeId;
		private int m_oldOutputPortId;
		public MoveOutputConnectionActionData( int inputNodeId, int inputPortId, int newOutputNodeId, int newOutputPortId, int oldOutputNodeId, int oldOutputPortId )
		{
			m_inputNodeId = inputNodeId;
			m_inputPortId = inputPortId;
			m_newOutputNodeId = newOutputNodeId;
```

```
                        m_newOutputPortId = newOutputPortId;
                        m_oldOutputNodeId = oldOutputNodeId;
                        m_oldOutputPortId = oldOutputPortId;
                }
                public override void ExecuteForward()
                {
                        UIUtils.DeleteConnection( false, m_oldOutputNodeId, m_oldOutputNodeId, false, true );
                        UIUtils.ConnectInputToOutput( m_inputNodeId, m_inputPortId, m_newOutputNodeId, m_newOutputPortId );
                }
                public override void ExecuteReverse()
                {
                        base.ExecuteReverse();
                        UIUtils.DeleteConnection( false, m_newOutputNodeId, m_newOutputPortId, false, true );
                        UIUtils.ConnectInputToOutput( m_inputNodeId, m_inputPortId, m_oldOutputNodeId, m_oldOutputPortId );
                }
                public override string ToString()
                {
                        return "Move Input Connection Node - Input Node: " + m_inputNodeId + " Input Port: " + m_inputPortId + " Old Output Node: " + m_oldOutputNodeId + " Old Output Port: " + m_oldOutputPortId + " New Output Node: " + m_newOutputNodeId + " New Output Port: " + m_newOutputPortId;
                }
        }
        public class CreateNewGraphActionData : ActionData
        {
                private string m_name;
                public CreateNewGraphActionData( string name )
                {
                        m_name = name;
                }
                public override void ExecuteForward()
                {
                        UIUtils.CreateNewGraph( m_name );
                }
        }
        public class ChangeNodePropertiesActionData : ActionData
        {
                private string m_originalProperties;
                private string m_newProperties;
                private int m_nodeId;
                public ChangeNodePropertiesActionData( ParentNode node, string originalProperties )
                {
                        m_nodeId = node.UniqueId;
                        m_originalProperties = originalProperties;
                        m_newProperties = string.Empty;
                        string trash = string.Empty;
                        node.WriteToString( ref m_newProperties, ref trash );
                }
                public ChangeNodePropertiesActionData( int nodeId, string originalProperties )
                {
                        m_nodeId = nodeId;
                        m_originalProperties = originalProperties;
                        m_newProperties = string.Empty;
                        string trash = string.Empty;
                        UIUtils.GetNode( nodeId ).WriteToString( ref m_newProperties, ref trash );
                }
                public override void ExecuteForward()
                {
                        string[] properties = m_newProperties.Split( IOUtils.FIELD_SEPARATOR );
                        UIUtils.GetNode( m_nodeId ).ReadFromString( ref properties );
                }
                public override void ExecuteReverse()
                {
                        string[] properties = m_originalProperties.Split( IOUtils.FIELD_SEPARATOR );
                        UIUtils.GetNode( m_nodeId ).ReadFromString( ref properties );
                }
                public override string ToString()
                {
                        return "Change Node Propertie - Node: " + m_nodeId + "\nOriginal Properties:\n" + m_originalProperties + "\nNew Properties:\n" + m_newProperties;
                }
        }
}
```

ActionData.cs.meta

```
fileFormatVersion: 2
guid: 29204f353101f46439a93f1c503d3197
timeCreated: 1481126954
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
```

ActionLog.cs

```
using System.Collections.Generic;
namespace AmplifyShaderEditor
{
        public class ActionLog
        {
                private int m_maxCount;
                private int m_index;
                private List<ActionData> m_sequence;
                public ActionLog(int maxCount)
                {
```

```
            m_maxCount = maxCount;
            m_index = 0;
            m_sequence = new List<ActionData>();
        }
        public void AddToLog(ActionData actionData)
        {
            if (m_sequence.Count > m_maxCount)
            {
                m_sequence.RemoveAt(0);
            }
            m_sequence.Add(actionData);
            m_index = m_sequence.Count - 1;
        }
        public void UndoLastAction()
        {
            if ( m_index > -1 && m_index < m_sequence.Count )
                m_sequence[m_index--].ExecuteReverse();
        }
        public void RedoLastAction()
        {
            if (m_index < (m_sequence.Count - 1))
                m_sequence[++m_index].ExecuteForward();
        }
        public void ClearLog()
        {
            m_sequence.Clear();
            m_index = 0;
        }
        public void Destroy()
        {
            m_sequence.Clear();
            m_sequence = null;
        }
    }
}
```

ActionLog.cs.meta

```
fileFormatVersion: 2
guid: bc089a69595d8994cb89946a919517c2
timeCreated: 1481126958
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
```

ActionSequence.cs

```
using System.Collections.Generic;
namespace AmplifyShaderEditor
{
    public class ActionSequence
    {
        private string m_name;
        private List<ActionData> m_sequence;
        public ActionSequence( string name )
        {
            m_name = name;
            m_sequence = new List<ActionData>();
        }
        public void AddToSequence( ActionData actionData )
        {
            m_sequence.Add( actionData );
        }
        public void Execute()
        {
            for ( int i = 0; i < m_sequence.Count; i++ )
            {
                m_sequence[ i ].ExecuteForward();
            }
        }
        public void Destroy()
        {
            m_sequence.Clear();
            m_sequence = null;
        }
        public string Name { get { return m_name; } }
    }
}
```

ActionSequence.cs.meta

```
fileFormatVersion: 2
guid: 43bd963fa46ee9c4680dacff1d8dc0b9
timeCreated: 1481126955
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
```

```
	Actions.meta
	fileFormatVersion: 2
guid: a2ba8588f45692f4ea2fa5afa9faf434
folderAsset: yes
timeCreated: 1481126944
licenseType: Store
DefaultImporter:
  userData:
  assetBundleName:
  assetBundleVariant:
	Constants.cs
	using UnityEngine;
using System.Collections.Generic;
namespace AmplifyShaderEditor
{
	public struct Constants
	{
		public readonly static string[] FaceMacros =
		{
			"#if defined(SHADER_API_GLCORE) || defined(SHADER_API_GLES) || defined(SHADER_API_GLES3) || defined(SHADER_API_D3D9)",
			"#define FRONT_FACE_SEMANTIC VFACE",
			"#define FRONT_FACE_TYPE float",
			"#else",
			"#define FRONT_FACE_SEMANTIC SV_IsFrontFace",
			"#define FRONT_FACE_TYPE bool",
			"#endif"
		};
		{
			"#if defined(SHADER_API_D3D11) || defined(SHADER_API_XBOXONE) || defined(UNITY_COMPILER_HLSLCC)//ASE Args Macros",
			"#define ASE_TEXTURE2D_ARGS(textureName) Texture2D textureName, SamplerState sampler##textureName",
			"#define ASE_TEXTURE3D_ARGS(textureName) Texture3D textureName, SamplerState sampler##textureName",
			"#define ASE_TEXTURECUBE_ARGS(textureName) TextureCube textureName, SamplerState sampler##textureName",
			"#define ASE_TEXTURE2D_PARAMS(textureName) textureName, sampler##textureName",
			"#define ASE_TEXTURE3D_PARAMS(textureName) textureName, sampler##textureName",
			"#define ASE_TEXTURECUBE_PARAMS(textureName) textureName, sampler##textureName",
			"#define ASE_TEXTURE2D_ARRAY_PARAMS(textureName) textureName, sampler##textureName",
			"#else//ASE Args Macros",
			"#define ASE_TEXTURE2D_ARGS(textureName) sampler2D textureName",
			"#define ASE_TEXTURE3D_ARGS(textureName) sampler3D textureName",
			"#define ASE_TEXTURECUBE_ARGS(textureName) samplerCUBE textureName",
			"#define ASE_TEXTURE2D_PARAMS(textureName) textureName",
			"#define ASE_TEXTURE3D_PARAMS(textureName) textureName",
			"#define ASE_TEXTURECUBE_PARAMS(textureName) textureName",
			"#define ASE_TEXTURE2D_ARRAY_PARAMS(textureName) textureName",
			"#endif//ASE Args Macros\n"
		};
		public readonly static string[] CustomASEDeclararionMacros =
		{
			"#define ASE_TEXTURE2D(textureName) {0}2D(textureName)",
			"#define ASE_TEXTURE2D_ARRAY(textureName) {0}2D_ARRAY(textureName)",
			"#define ASE_TEXTURE3D(textureName) {0}3D(textureName)",
			"#define ASE_TEXTURECUBE(textureName) {0}CUBE(textureName)\n"
		};
		public readonly static string[] CustomASEStandarSamplingMacrosHelper =
		{
			"#if defined(SHADER_API_D3D11) || defined(SHADER_API_XBOXONE) || defined(UNITY_COMPILER_HLSLCC)//ASE Sampling Macros",
			"#else//ASE Sampling Macros",
			"#endif//ASE Sampling Macros\n"
		};*/
		{
			"#define ASE_SAMPLE_TEXTURE2D(textureName,{0}coords) {1}2D{2}(textureName,{0}coords)",
			"#define ASE_SAMPLE_TEXTURE2D_LOD(textureName, {0}coord2, lod) {1}2D{2}_LOD(textureName, {0}coord2, lod)",
			"#define ASE_SAMPLE_TEXTURE2D_BIAS(textureName,{0}coord2, bias) {1}2D{2}_BIAS(textureName,{0}coord2, bias)",
			"#define ASE_SAMPLE_TEXTURE2D_GRAD(textureName,{0}coord2, dpdx, dpdy) {1}2D{2}_GRAD(textureName,{0}coord2, dpdx, dpdy)",
			"#define ASE_SAMPLE_TEXTURE3D(textureName,{0}coord3) {1}3D{2}(textureName,{0}coord3)",
			"#define ASE_SAMPLE_TEXTURE3D_LOD(textureName,{0}coord3, lod) {1}3D{2}_LOD(textureName,{0}coord3, lod)",
			"#define ASE_SAMPLE_TEXTURE3D_BIAS(textureName,{0}coord3, bias) {1}3D{2}_BIAS(textureName,{0}coord3, bias)",
			"#define ASE_SAMPLE_TEXTURE3D_GRAD(textureName,{0}coord3, dpdx, dpdy) {1}3D{2}_GRAD(textureName,{0}coord3, dpdx, dpdy)",
			"#define ASE_SAMPLE_TEXTURECUBE(textureName,{0}coord3) {1}CUBE{2}(textureName,{0}coord3)",
			"#define ASE_SAMPLE_TEXTURECUBE_LOD(textureName,{0}coord3, lod) {1}CUBE{2}_LOD(textureName,{0}coord3, lod)",
			"#define ASE_SAMPLE_TEXTURECUBE_BIAS(textureName,{0}coord3, bias) {1}CUBE{2}_BIAS(textureName,{0}coord3, bias)\n"
		};*/
		{
			"#define ASE_TEXTURE2D_ARGS(textureName) TEXTURE2D(textureName), SAMPLER(textureName)",
			"#define ASE_TEXTURE3D_ARGS(textureName) TEXTURE3D(textureName), SAMPLER(textureName)",
			"#define ASE_TEXTURECUBE_ARGS(textureName) TEXTURECUBE(textureName), SAMPLER(textureName)",
			"#define ASE_TEXTURE2D_PARAMS(textureName) textureName, sampler##textureName",
			"#define ASE_TEXTURE3D_PARAMS(textureName) textureName, sampler##textureName",
			"#define ASE_TEXTURECUBE_PARAMS(textureName) textureName, sampler##textureName",
			"#define ASE_TEXTURE2D_ARRAY_PARAMS(textureName) textureName, sampler##textureName\n"
		};*/
		public readonly static RenderTextureFormat PreviewFormat = RenderTextureFormat.ARGBFloat;
		public readonly static int PreviewSize = 128;
		public readonly static List<string> UnityNativeInspectors = new List<string>
		{
			"Rendering.HighDefinition.LightingShaderGraphGUI",
			"Rendering.HighDefinition.HDUnlitGUI",
			"UnityEditor.Rendering.HighDefinition.HDLitGUI",
			"UnityEditor.ShaderGraph.PBRMasterGUI",
			"UnityEditor.Rendering.HighDefinition.DecalGUI",
			"UnityEditor.Rendering.HighDefinition.FabricGUI",
			"UnityEditor.Experimental.Rendering.HDPipeline.HDLitGUI",
			"Rendering.HighDefinition.DecalGUI",
```

```
                "Rendering.HighDefinition.LitShaderGraphGUI",
                "Rendering.HighDefinition.DecalShaderGraphGUI",
                "UnityEditor.ShaderGraphUnlitGUI",
                "UnityEditor.ShaderGraphLitGUI",
                "UnityEditor.Rendering.Universal.DecalShaderGraphGUI"
        };
        public readonly static Dictionary<string, string> CustomInspectorHD7To10 = new Dictionary<string, string>
        {
                { "UnityEditor.Rendering.HighDefinition.DecalGUI","Rendering.HighDefinition.DecalGUI"},
                { "UnityEditor.Rendering.HighDefinition.FabricGUI","Rendering.HighDefinition.LightingShaderGraphGUI"},
                { "UnityEditor.Rendering.HighDefinition.HDLitGUI","Rendering.HighDefinition.LitShaderGraphGUI"},
                { "UnityEditor.Experimental.Rendering.HDPipeline.HDLitGUI","Rendering.HighDefinition.LitShaderGraphGUI"},
        };
        public readonly static Dictionary<string , string> CustomInspectorURP10To12 = new Dictionary<string , string>
        {
                { "UnityEditor.ShaderGraph.PBRMasterGUI","UnityEditor.ShaderGraphLitGUI"},
        };
        public readonly static Dictionary<string , string> CustomInspectorHDLegacyTo11 = new Dictionary<string , string>
        {
                { "UnityEditor.Rendering.HighDefinition.DecalGUI","Rendering.HighDefinition.DecalShaderGraphGUI"},
                { "Rendering.HighDefinition.DecalGUI","Rendering.HighDefinition.DecalShaderGraphGUI"},
                { "UnityEditor.Rendering.HighDefinition.FabricGUI","Rendering.HighDefinition.LightingShaderGraphGUI"},
                { "UnityEditor.Rendering.HighDefinition.HDLitGUI","Rendering.HighDefinition.LitShaderGraphGUI"},
                { "UnityEditor.Experimental.Rendering.HDPipeline.HDLitGUI","Rendering.HighDefinition.LitShaderGraphGUI"},
        };
        public readonly static string CustomASEStandardSamplerParams = "#define ASE_TEXTURE_PARAMS(textureName) textureName\n";
        public readonly static string[] CustomASESRPTextureArrayMacros =
        {
                "#define ASE_TEXTURE2D_ARRAY_ARGS(textureName) TEXTURE2D_ARRAY_ARGS(textureName,sampler##textureName)\n" ,
                "#define ASE_TEXTURE2D_ARRAY_PARAM(textureName) TEXTURE2D_ARRAY_PARAM(textureName,sampler##textureName)\n" ,
                "#define ASE_SAMPLE_TEXTURE2D_ARRAY(textureName, coord3) textureName.Sample(sampler##textureName, coord3)",
                "#define ASE_SAMPLE_TEXTURE2D_ARRAY_LOD(textureName, coord3, lod) textureName.SampleLevel(sampler##textureName, coord3, lod)"
        };
        public readonly static string CustomASESRPSamplerParams = "#define ASE_TEXTURE_PARAMS(textureName) textureName, sampler##textureName\n";
        public readonly static string[] CustomSRPSamplingMacros =
        {
                "#if defined(SHADER_API_D3D11) || defined(SHADER_API_XBOXONE) || defined(UNITY_COMPILER_HLSLCC) || defined(SHADER_API_PSSL) || (defined(SHADER_TARGET_SURFACE_ANALYSIS) && !defined(SHADER_TARGET_SURFACE_ANALYSIS_MOJOSHADER))//3D SRP MACROS",
                "#define SAMPLE_TEXTURE3D_GRAD(textureName, samplerName, coord3, dpdx, dpdy) textureName.SampleGrad(samplerName, coord3, dpdx, dpdy)",
                "#define SAMPLE_TEXTURE3D_BIAS(textureName, samplerName, coord3, bias) textureName.SampleBias(samplerName, coord3, bias)",
                "#else//3D SRP MACROS",
                "#define SAMPLE_TEXTURE3D_GRAD(textureName, samplerName, coord3, dpdx, dpdy) SAMPLE_TEXTURE3D(textureName, samplerName, coord3)",
                "#define SAMPLE_TEXTURE3D_BIAS(textureName, samplerName, coord3, bias) SAMPLE_TEXTURE3D(textureName, samplerName, coord3)",
                "#endif//3D SRP MACROS\n"
        };
        public readonly static Dictionary<TextureType, string> TexDeclarationSRPMacros = new Dictionary<TextureType, string>
        {
                { TextureType.Texture2D,"TEXTURE2D({0}); SAMPLER(sampler{0});"},
                { TextureType.Texture3D,"TEXTURE3D({0}); SAMPLER(sampler{0});"},
                { TextureType.Cube,"TEXTURECUBE({0}); SAMPLER(sampler{0});"},
                { TextureType.Texture2DArray,"TEXTURE2D_ARRAY({0}); SAMPLER(sampler{0});"},
        };
        public readonly static Dictionary<TextureType, string> SamplerDeclarationSRPMacros = new Dictionary<TextureType, string>
        {
                { TextureType.Texture2D,"SAMPLER(sampler{0});"},
                { TextureType.Texture3D,"SAMPLER(sampler{0});"},
                { TextureType.Cube,"SAMPLER(sampler{0});"},
                { TextureType.Texture2DArray,"SAMPLER(sampler{0});"},
        };
        public readonly static Dictionary<TextureType, string> TexDeclarationNoSamplerSRPMacros = new Dictionary<TextureType, string>
        {
                { TextureType.Texture2D,"TEXTURE2D({0})"},
                { TextureType.Texture3D,"TEXTURE3D({0})"},
                { TextureType.Cube,"TEXTURECUBE({0})"},
                { TextureType.Texture2DArray,"TEXTURE2D_ARRAY({0})"},
        };
        public readonly static Dictionary<TextureType, string> TexSampleSRPMacros = new Dictionary<TextureType, string>
        {
                { TextureType.Texture2D,"SAMPLE_TEXTURE2D(0)( {1}, {2}, {3} )"},
                { TextureType.Texture3D,"SAMPLE_TEXTURE3D(0)( {1}, {2}, {3} )"},
                { TextureType.Cube,"SAMPLE_TEXTURECUBE{0}( {1}, {2}, {3} )"},
                { TextureType.Texture2DArray,"SAMPLE_TEXTURE2D_ARRAY{0}( {1}, {2}, {3} )"},
        };
        public readonly static Dictionary<TextureType, string> TexParams = new Dictionary<TextureType, string>
        {
                { TextureType.Texture2D,"ASE_TEXTURE2D_PARAMS({0})"},
                { TextureType.Texture3D,"ASE_TEXTURE3D_PARAMS({0})"},
                { TextureType.Cube,"ASE_TEXTURECUBE_PARAMS({0})"},
                { TextureType.Texture2DArray,"ASE_TEXTURE2D_ARRAY_PARAMS({0})"},
        };
        public readonly static Dictionary<WirePortDataType, TextureType> WireToTexture = new Dictionary<WirePortDataType, TextureType>
        {
                { WirePortDataType.SAMPLER1D,TextureType.Texture1D},
                { WirePortDataType.SAMPLER2D,TextureType.Texture2D},
                { WirePortDataType.SAMPLER3D,TextureType.Texture3D},
                { WirePortDataType.SAMPLERCUBE,TextureType.Cube},
                { WirePortDataType.SAMPLER2DARRAY,TextureType.Texture2DArray},
        };
        public readonly static Dictionary<TextureType, WirePortDataType> TextureToWire = new Dictionary<TextureType, WirePortDataType>
        {
                { TextureType.Texture1D,WirePortDataType.SAMPLER1D},
                { TextureType.Texture2D,WirePortDataType.SAMPLER2D},
                { TextureType.Texture3D, WirePortDataType.SAMPLER3D},
                { TextureType.Cube,WirePortDataType.SAMPLERCUBE},
```

```
        { TextureType.Texture2DArray,WirePortDataType.SAMPLER2DARRAY},
        { TextureType.ProceduralTexture,WirePortDataType.SAMPLER2D},
    };
    public readonly static string SamplingMacrosDirective = "#define ASE_USING_SAMPLING_MACROS 1";
    public readonly static string[] CustomASEStandarSamplingMacrosHelper =
    {
        "#if defined(SHADER_API_D3D11) || defined(SHADER_API_XBOXONE) || defined(UNITY_COMPILER_HLSLCC) || defined(SHADER_API_PSSL) || (defined(SHADER_TARGET_SURFACE_ANALYSIS) && !defined(SHADER_TARGET_SURFACE_ANALYSIS_MOJOSHADER))//ASE Sampler Macros",
        "#if defined(SHADER_API_D3D11) || defined(SHADER_API_XBOXONE) || defined(UNITY_COMPILER_HLSLCC) || defined(SHADER_API_PSSL)//ASE Sampler Macros",
        "#else//ASE Sampling Macros",
        "#endif//ASE Sampling Macros\n"
    };
    public readonly static string[] CustomASEArraySamplingMacrosRecent =
    {
        "#define UNITY_SAMPLE_TEX2DARRAY(tex,coord) tex.Sample(sampler##tex,coord)",
        "#define UNITY_SAMPLE_TEX2DARRAY_LOD(tex,coord,lod) tex.SampleLevel(sampler##tex,coord, lod)",
        "#define UNITY_SAMPLE_TEX2DARRAY_BIAS(tex,coord,bias) tex.SampleBias(sampler##tex,coord,bias)",
        "#define UNITY_SAMPLE_TEX2DARRAY_GRAD(tex,coord,ddx,ddy) tex.SampleGrad(sampler##tex,coord,ddx,ddy)",
    };
    public readonly static string[] CustomASEArraySamplingMacrosOlder =
    {
        "#define UNITY_SAMPLE_TEX2DARRAY(tex,coord) tex2DArray(tex,coord)",
        "#define UNITY_SAMPLE_TEX2DARRAY_LOD(tex,coord,lod) tex2DArraylod(tex, float4(coord,lod))",
        "#define UNITY_SAMPLE_TEX2DARRAY_BIAS(tex,coord,bias) tex2DArray(tex,coord)",
        "#define UNITY_SAMPLE_TEX2DARRAY_GRAD(tex,coord,ddx,ddy) tex2DArray(tex,coord)",
    };
    public readonly static string[] CustomASEStandarSamplingMacrosRecent =
    {
        "#define SAMPLE_TEXTURE2D(tex,samplerTex,coord) tex.Sample(samplerTex,coord)",
        "#define SAMPLE_TEXTURE2D_LOD(tex,samplerTex,coord,lod) tex.SampleLevel(samplerTex,coord, lod)",
        "#define SAMPLE_TEXTURE2D_BIAS(tex,samplerTex,coord,bias) tex.SampleBias(samplerTex,coord,bias)",
        "#define SAMPLE_TEXTURE2D_GRAD(tex,samplerTex,coord,ddx,ddy) tex.SampleGrad(samplerTex,coord,ddx,ddy)",
        "#define SAMPLE_TEXTURE3D(tex,samplerTex,coord) tex.Sample(samplerTex,coord)",
        "#define SAMPLE_TEXTURE3D_LOD(tex,samplerTex,coord,lod) tex.SampleLevel(samplerTex,coord, lod)",
        "#define SAMPLE_TEXTURE3D_BIAS(tex,samplerTex,coord,bias) tex.SampleBias(samplerTex,coord,bias)",
        "#define SAMPLE_TEXTURE3D_GRAD(tex,samplerTex,coord,ddx,ddy) tex.SampleGrad(samplerTex,coord,ddx,ddy)",
        "#define SAMPLE_TEXTURECUBE(tex,samplerTex,coord) tex.Sample(samplerTex,coord)",
        "#define SAMPLE_TEXTURECUBE_LOD(tex,samplerTex,coord,lod) tex.SampleLevel(samplerTex,coord, lod)",
        "#define SAMPLE_TEXTURECUBE_BIAS(tex,samplerTex,coord,bias) tex.SampleBias(samplerTex,coord,bias)",
        "#define SAMPLE_TEXTURECUBE_GRAD(tex,samplerTex,coord,ddx,ddy) tex.SampleGrad(samplerTex,coord,ddx,ddy)",
        "#define SAMPLE_TEXTURE2D_ARRAY(tex,samplerTex,coord) tex.Sample(samplerTex,coord)",
        "#define SAMPLE_TEXTURE2D_ARRAY_LOD(tex,samplerTex,coord,lod) tex.SampleLevel(samplerTex,coord, lod)",
        "#define SAMPLE_TEXTURE2D_ARRAY_BIAS(tex,samplerTex,coord,bias) tex.SampleBias(samplerTex,coord,bias)",
        "#define SAMPLE_TEXTURE2D_ARRAY_GRAD(tex,samplerTex,coord,ddx,ddy) tex.SampleGrad(samplerTex,coord,ddx,ddy)",
    };
    public readonly static string[] CustomASEStandarSamplingMacrosOlder =
    {
        "#define SAMPLE_TEXTURE2D(tex,samplerTex,coord) tex2D(tex,coord)",
        "#define SAMPLE_TEXTURE2D_LOD(tex,samplerTex,coord,lod) tex2Dlod(tex,float4(coord,0,lod))",
        "#define SAMPLE_TEXTURE2D_BIAS(tex,samplerTex,coord,bias) tex2Dbias(tex,float4(coord,0,bias))",
        "#define SAMPLE_TEXTURE2D_GRAD(tex,samplerTex,coord,ddx,ddy) tex2Dgrad(tex,coord,ddx,ddy)",
        "#define SAMPLE_TEXTURE3D(tex,samplerTex,coord) tex3D(tex,coord)",
        "#define SAMPLE_TEXTURE3D_LOD(tex,samplerTex,coord,lod) tex3Dlod(tex,float4(coord,lod))",
        "#define SAMPLE_TEXTURE3D_BIAS(tex,samplerTex,coord,bias) tex3D(tex,coord)",
        "#define SAMPLE_TEXTURE3D_GRAD(tex,samplerTex,coord,ddx,ddy) tex3D(tex,coord)",
        "#define SAMPLE_TEXTURECUBE(tex,samplertex,coord) texCUBE(tex,coord)",
        "#define SAMPLE_TEXTURECUBE_LOD(tex,samplertex,coord,lod) texCUBElod (tex,half4(coord,lod))",
        "#define SAMPLE_TEXTURECUBE_BIAS(tex,samplertex,coord,bias) texCUBE(tex,coord)",
        "#define SAMPLE_TEXTURECUBE_GRAD(tex,samplertex,coord,ddx,ddy) texCUBE(tex,coord)",
        "#define SAMPLE_TEXTURE2D_ARRAY(tex,samplertex,coord) tex2DArray(tex,coord)",
        "#define SAMPLE_TEXTURE2D_ARRAY_LOD(tex,samplertex,coord,lod) tex2DArraylod(tex, float4(coord,lod))",
        "#define SAMPLE_TEXTURE2D_ARRAY_BIAS(tex,samplerTex,coord,bias) tex2DArray(tex,coord)",
        "#define SAMPLE_TEXTURE2D_ARRAY_GRAD(tex,samplerTex,coord,ddx,ddy) tex2DArray(tex,coord)",
    };
    public readonly static string[] CustomArraySamplingMacros =
    {
        "#if defined(UNITY_COMPILER_HLSL2GLSL) || defined(SHADER_TARGET_SURFACE_ANALYSIS)//ASE Array Sampler Macros",
        "#define ASE_SAMPLE_TEX2DARRAY_GRAD(tex,coord,dx,dy) UNITY_SAMPLE_TEX2DARRAY (tex,coord)",
        "#else//ASE Array Sampler Macros",
        "#define ASE_SAMPLE_TEX2DARRAY_GRAD(tex,coord,dx,dy) tex.SampleGrad (sampler##tex,coord,dx,dy)",
        "#endif//ASE Array Sampler Macros\n"
    };
    public readonly static Dictionary<TextureType, string> TexDeclarationStandardMacros = new Dictionary<TextureType, string>
    {
        { TextureType.Texture2D,"UNITY_DECLARE_TEX2D({0});"},
        { TextureType.Texture3D,"UNITY_DECLARE_TEX3D({0});"},
        { TextureType.Cube,"UNITY_DECLARE_TEXCUBE({0});"},
        { TextureType.Texture2DArray,"UNITY_DECLARE_TEX2DARRAY({0});"}
    };
    public readonly static Dictionary<TextureType, string> TexDeclarationNoSamplerStandardMacros = new Dictionary<TextureType, string>
    {
        { TextureType.Texture2D,"UNITY_DECLARE_TEX2D_NOSAMPLER({0})"},
        { TextureType.Texture3D,"UNITY_DECLARE_TEX3D_NOSAMPLER({0})"},
        { TextureType.Cube,"UNITY_DECLARE_TEXCUBE_NOSAMPLER({0})"},
        { TextureType.Texture2DArray,"UNITY_DECLARE_TEX2DARRAY_NOSAMPLER({0})"}
    };
    public readonly static Dictionary<TextureType, string> TexSampleStandardMacros = new Dictionary<TextureType, string>
    {
        { TextureType.Texture2D,"UNITY_SAMPLE_TEX2D(0)( {1}, {3} )"},
        { TextureType.Texture3D,"UNITY_SAMPLE_TEX3D(0)( {1}, {3} )"},
        { TextureType.Cube,"UNITY_SAMPLE_TEXCUBE(0)( {1}, {3} )"},
        { TextureType.Texture2DArray,"UNITY_SAMPLE_TEX2DARRAY(0)( {1}, {3} )"}
    };
    public readonly static Dictionary<TextureType, string> TexSampleSamplerStandardMacros = new Dictionary<TextureType, string>
```

```
{
		{ TextureType.Texture2D,"SAMPLE_TEXTURE2D(0)( {1}, {2}, {3} )"},
		{ TextureType.Texture3D,"SAMPLE_TEXTURE3D(0)( {1}, {2}, {3} )"},
		{ TextureType.Cube,"SAMPLE_TEXTURECUBE(0)( {1}, {2}, {3} )"},
		{ TextureType.Texture2DArray,"SAMPLE_TEXTURE2D_ARRAY(0)( {1}, {2}, {3} )"}
};
public readonly static Dictionary<TextureType, string> TexSampleStandard = new Dictionary<TextureType, string>
{
		{ TextureType.Texture2D,"tex2D(0)( {1}, {2} )"},
		{ TextureType.Texture3D,"tex3D(0)( {1}, {2} )"},
		{ TextureType.Cube,"texCUBE(0)( {1}, {2} )"},
		{ TextureType.Texture2DArray,"tex2DArray(0)( {1}, {2} )"}
};
public readonly static char LineFeedSeparator = '$';
public readonly static char SemiColonSeparator = '@';
public readonly static string AppDataFullName = "appdata_full";
public readonly static string CustomAppDataFullName = "appdata_full_custom";
public readonly static string CustomAppDataFullBody =
"\n\t\tstruct appdata_full_custom\n" +
"\t\t{\n" +
"\t\t\tfloat4 vertex : POSITION;\n" +
"\t\t\tfloat4 tangent : TANGENT;\n" +
"\t\t\tfloat3 normal : NORMAL;\n" +
"\t\t\tfloat4 texcoord : TEXCOORD0;\n" +
"\t\t\tfloat4 texcoord1 : TEXCOORD1;\n" +
"\t\t\tfloat4 texcoord2 : TEXCOORD2;\n" +
"\t\t\tfloat4 texcoord3 : TEXCOORD3;\n" +
"\t\t\tfloat4 color : COLOR;\n" +
"\t\t\tUNITY_VERTEX_INPUT_INSTANCE_ID\n";
public readonly static string IncludeFormat = "#include \"{0}\"";
public readonly static string PragmaFormat = "#pragma {0}";
public readonly static string DefineFormat = "#define {0}";
public readonly static string RenderTypeHelperStr = "RenderType";
public readonly static string RenderQueueHelperStr = "Queue";
public readonly static string DisableBatchingHelperStr = "DisableBatching";
public readonly static string DefaultShaderName = "New Amplify Shader";
public readonly static string UndoReplaceMasterNodeId = "Replacing Master Node";
public readonly static string UnityLightingLib = "Lighting.cginc";
public readonly static string UnityAutoLightLib = "AutoLight.cginc";
public readonly static string UnityBRDFLib = "UnityStandardBRDF.cginc";
public readonly static string LocalValueDecWithoutIdent = "{0} {1} = {2};";
public readonly static string CustomTypeLocalValueDecWithoutIdent = "{0} {1} ={{0}}{2};";
public readonly static string LocalValueDefWithoutIdent = "{0} {1} {2};";
public readonly static string TilingOffsetFormat = "{0} * {1} + {2}";
public static string InvalidPostProcessDatapath = "__DELETED_GUID_Trash";
public static float PlusMinusButtonLayoutWidth = 15;
public static float NodeButtonSizeX = 16;
public static float NodeButtonSizeY = 16;
public static float NodeButtonDeltaX = 5;
public static float NodeButtonDeltaY = 11;
public readonly static string SafeNormalizeInfoStr = "With Safe Normalize division by 0 is prevented over the normalize operation at the expense of additional instructions on shader.";
public readonly static string ReservedPropertyNameStr = "Property name '{0}' is reserved and cannot be used";
public readonly static string NumericPropertyNameStr = "Property name '{0}' is numeric thus cannot be used";
public readonly static string DeprecatedMessageStr = "Node '{0}' is deprecated. Use node '{1}' instead.";
public readonly static string DeprecatedNoAlternativeMessageStr = "Node '{0}' is deprecated and should be removed.";
public readonly static string UndoChangePropertyTypeNodesId = "Changing Property Types";
public readonly static string UndoChangeTypeNodesId = "Changing Nodes Types";
public readonly static string UndoMoveNodesId = "Moving Nodes";
public readonly static string UndoRegisterFullGraphId = "Register Graph";
public readonly static string UndoAddNodeToCommentaryId = "Add node to Commentary";
public readonly static string UndoRemoveNodeFromCommentaryId = "Remove node from Commentary";
public readonly static string UndoCreateDynamicPortId = "Create Dynamic Port";
public readonly static string UndoDeleteDynamicPortId = "Destroy Dynamic Port";
public readonly static string UndoRegisterNodeId = "Register Object";
public readonly static string UndoUnregisterNodeId = "Unregister Object";
public readonly static string UndoCreateNodeId = "Create Object";
public readonly static string UndoPasteNodeId = "Paste Object";
public readonly static string UndoDeleteNodeId = "Destroy Object";
public readonly static string UndoDeleteConnectionId = "Destroy Connection";
public readonly static string UndoCreateConnectionId = "Create Connection";
public readonly static float MenuDragSpeed = -0.5f;
public readonly static string DefaultCustomInspector = "ASEMaterialInspector";
public readonly static string ReferenceTypeStr = "Mode";
public readonly static string AvailableReferenceStr = "Reference";
public readonly static string InstancePostfixStr = " (Reference) ";
public readonly static string ASEMenuName = "Amplify Shader";
public readonly static string LodCrossFadeOption2017 = "dithercrossfade";
public readonly static string UnityShaderVariables = "UnityShaderVariables.cginc";
public readonly static string UnityCgLibFuncs = "UnityCG.cginc";
public readonly static string UnityStandardUtilsLibFuncs = "UnityStandardUtils.cginc";
public readonly static string UnityPBSLightingLib = "UnityPBSLighting.cginc";
public readonly static string UnityDeferredLightLib = "UnityDeferredLibrary.cginc";
public readonly static string ATSharedLibGUID = "ba242738c4be3324aa88d126f7cc19f9";
public readonly static string CameraDepthTextureValue = "UNITY_DECLARE_DEPTH_TEXTURE( _CameraDepthTexture );";
public readonly static string CameraDepthTextureValue = "uniform sampler2D _CameraDepthTexture;";
public readonly static string CameraDepthTextureLWEnabler = "REQUIRE_DEPTH_TEXTURE 1";
public readonly static string CameraDepthTextureTexelSize = "uniform float4 _CameraDepthTexture_TexelSize;";
public readonly static string InstanceIdMacro = "UNITY_VERTEX_INPUT_INSTANCE_ID";
public readonly static string InstanceIdVariable = "UNITY_GET_INSTANCE_ID({0})";
public readonly static string HelpURL = "http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor";
public readonly static string NodeCommonUrl = "http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/";
public readonly static string CommunityNodeCommonUrl = "http://wiki.amplify.pt/index.php?title=Unity_Products:Amplify_Shader_Editor/";
public readonly static Color InfiniteLoopColor = Color.red;
```

```
public readonly static Color DefaultCategoryColor = new Color( 0.26f, 0.35f, 0.44f, 1.0f );
public readonly static Color NodeBodyColor = new Color( 1f, 1f, 1f, 1.0f );
public readonly static Color ModeTextColor = new Color( 1f, 1f, 1f, 0.25f );
public readonly static Color ModeIconColor = new Color( 1f, 1f, 1f, 0.75f );
public readonly static Color PortTextColor = new Color( 1f, 1f, 1f, 0.5f );
public readonly static Color PortLockedTextColor = new Color( 1f, 1f, 1f, 0.35f );
public readonly static Color BoxSelectionColor = new Color( 0.5f, 0.75f, 1f, 0.33f );
public readonly static Color SpecialRegisterLocalVarSelectionColor = new Color( 0.27f, 0.52f, 1.0f, 1f );
public readonly static Color SpecialGetLocalVarSelectionColor = new Color( 0.2f, 0.8f, 0.4f, 1f );
public readonly static Color NodeSelectedColor = new Color( 0.85f, 0.56f, 0f, 1f );
public readonly static Color NodeDefaultColor = new Color( 1f, 1f, 1f, 1f );
public readonly static Color NodeConnectedColor = new Color( 1.0f, 1f, 0.0f, 1f );
public readonly static Color NodeErrorColor = new Color( 1f, 0.5f, 0.5f, 1f );
public readonly static string NoSpecifiedCategoryStr = "<None>";
public readonly static int MINIMIZE_WINDOW_LOCK_SIZE = 630;
public readonly static int FoldoutMouseId = 0; // Left Mouse Button
public readonly static float SNAP_SQR_DIST = 200f;
public readonly static int INVALID_NODE_ID = -1;
public readonly static float WIRE_WIDTH = 7f;
public readonly static float WIRE_CONTROL_POINT_DIST = 0.7f;
public readonly static float WIRE_CONTROL_POINT_DIST_INV = 1.7f;
public readonly static float IconsLeftRightMargin = 5f;
public readonly static float PropertyPickerWidth = 16f;
public readonly static float PropertyPickerHeight = 16f;
public readonly static float PreviewExpanderWidth = 16f;
public readonly static float PreviewExpanderHeight = 16f;
public readonly static float TextFieldFontSize = 11f;
public readonly static float DefaultFontSize = 15f;
public readonly static float DefaultTitleFontSize = 13f;
public readonly static float PropertiesTitleFontSize = 11f;
public readonly static float MessageFontSize = 40f;
public readonly static float SelectedObjectFontSize = 30f;
public readonly static float PORT_X_ADJUST = 10;
public readonly static float PORT_INITIAL_X = 10;
public readonly static float PORT_INITIAL_Y = 40;
public readonly static float INPUT_PORT_DELTA_Y = 5;
public readonly static float PORT_TO_LABEL_SPACE_X = 5;
public readonly static float NODE_HEADER_HEIGHT = 32;
public readonly static float NODE_HEADER_EXTRA_HEIGHT = 5;
public readonly static float NODE_HEADER_LEFTRIGHT_MARGIN = 10;
public readonly static float MULTIPLE_SELECION_BOX_ALPHA = 0.5f;
public readonly static float RMB_CLICK_DELTA_TIME = 0.1f;
public readonly static float RMB_SCREEN_DIST = 10f;
public readonly static float CAMERA_MAX_ZOOM = 2f;
public readonly static float CAMERA_MIN_ZOOM = 1f;
public readonly static float CAMERA_ZOOM_SPEED = 0.1f;
public readonly static float ALT_CAMERA_ZOOM_SPEED = -0.05f;
public readonly static object INVALID_VALUE = null;
public readonly static float HORIZONTAL_TANGENT_SIZE = 100f;
public readonly static float OUTSIDE_WIRE_MARGIN = 5f;
public readonly static string SubTitleNameFormatStr = "Name( {0} )";
public readonly static string SubTitleSpaceFormatStr = "Space( {0} )";
public readonly static string SubTitleTypeFormatStr = "Type( {0} )";
public readonly static string SubTitleValueFormatStr = "Value( {0} )";
public readonly static string SubTitleConstFormatStr = "Const( {0} )";
public readonly static string SubTitleVarNameFormatStr = "Var( {0} )";
public readonly static string SubTitleRefNameFormatStr = "Ref( {0} )";
public readonly static string CodeWrapper = "{ {0} }";
public readonly static string InlineCodeWrapper = "{{\n{0}\n}}";
public readonly static string NodesDumpFormat = "{0};,{1},{2}\n";
public readonly static string TagFormat = " \"{0}\" = \"{1}\"";
public readonly static string LocalVarIdentation = "\t\t\t";
public readonly static string SimpleLocalValueDec = LocalVarIdentation + "{0} {1};\n";
public readonly static string LocalValueDec = LocalVarIdentation + LocalValueDecWithoutIdent + '\n';
public readonly static string LocalValueDef = LocalVarIdentation + "{0} = {1};\n";
public readonly static string CastHelper = "({0}).{1}";
public readonly static string PropertyLocalVarDec = "{0} {1} = {0}({2});";
public readonly static string[] UniformDec = { "uniform {0} {1};", "{0} {1};" };
public readonly static string PropertyValueLabel = "Value( {0} )";
public readonly static string ConstantsValueLabel = "Const( {0} )";
public readonly static string PropertyFloatFormatLabel = "0.##";
public readonly static string PropertyBigFloatFormatLabel = "0.###e+0";
public readonly static string PropertyIntFormatLabel = "0";
public readonly static string PropertyBigIntFormatLabel = "0e+0";
public readonly static string PropertyVectorFormatLabel = "0.##";
public readonly static string PropertyBigVectorFormatLabel = "0.##e+0";
public readonly static string PropertyMatrixFormatLabel = "0.#";
public readonly static string PropertyBigMatrixFormatLabel = "0.#e+0";
public readonly static string NoPropertiesLabel = "No assigned properties";
public readonly static string ValueLabel = "Value";
public readonly static string DefaultValueLabel = "Default Value";
public readonly static string MaterialValueLabel = "Material Value";
public readonly static GUIContent DefaultValueLabelContent = new GUIContent( "Default Value" );
public readonly static GUIContent MaterialValueLabelContent = new GUIContent( "Material Value" );
public readonly static string InputVarStr = "i";//"input";
public readonly static string OutputVarStr = "o";//"output";
public readonly static string CustomLightOutputVarStr = "s";
public readonly static string CustomLightStructStr = "Custom";
public readonly static string VertexShaderOutputStr = "o";
public readonly static string VertexShaderInputStr = "v";//"vertexData";
public readonly static string VertexDataFunc = "vertexDataFunc";
public readonly static string VirtualCoordNameStr = "vcoord";
public readonly static string VertexVecNameStr = "vertexVec";
```

```
public readonly static string VertexVecDecStr = "float3 " + VertexVecNameStr;
public readonly static string VertexVecVertStr = VertexShaderOutputStr + "." + VertexVecNameStr;
public readonly static string NormalVecNameStr = "normalVec";
public readonly static string NormalVecDecStr = "float3 " + NormalVecNameStr;
public readonly static string NormalVecFragStr = InputVarStr + "." + NormalVecNameStr;
public readonly static string NormalVecVertStr = VertexShaderOutputStr + "." + NormalVecNameStr;
public readonly static string IncidentVecNameStr = "incidentVec";
public readonly static string IncidentVecDecStr = "float3 " + IncidentVecNameStr;
public readonly static string IncidentVecDefStr = VertexShaderOutputStr + "." + IncidentVecNameStr + " = normalize( " + VertexVecNameStr + " - _WorldSpaceCameraPos.xyz)";
public readonly static string IncidentVecFragStr = InputVarStr + "." + IncidentVecNameStr;
public readonly static string IncidentVecVertStr = VertexShaderOutputStr + "." + IncidentVecNameStr;
public readonly static string WorldNormalLocalDecStr = "WorldNormalVector( " + Constants.InputVarStr + " , {0}{ 0,0,1 })";
public readonly static string VFaceVariable = "ASEVFace";
public readonly static string VFaceInput = "half ASEVFace : VFACE";
public readonly static string ColorVariable = "vertexColor";
public readonly static string ColorInput = "float4 vertexColor : COLOR";
public readonly static string NoStringValue = "None";
public readonly static string EmptyPortValue = "    ";
public readonly static string[] OverallInvalidChars = { "\r", "\n", "\\", " ", "-", ">", ".", "<", "\"", "\'", ":", "-", "[", "{", "!", "}", "]",",", "|", "=", "+", "`", "~", "/", "?", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "-" };
public readonly static string[] ShaderInvalidChars = { "\r", "\n", "\\", "\"", "\'", };
public readonly static string[] EnumInvalidChars = { "\r", "\n", "\\", "-", ">", " ", ",", "<", "\"", "\'", ";", "-", "[", "{", "]", "}", "=", "+", "`", "~", "/", "?", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "-" };
public readonly static string[] AttrInvalidChars = { "\r", "\n", "\\", ">", "<", "\"", "\'", ":", "-", "[", "{", "]", "}", "=", "+", "`", "~", "/", "?", "!", "@", "#", "$", "%", "^", "&", "*" };
public readonly static string[] HeaderInvalidChars = { "\r", "\n", "\\", ">", ",", "<", "\"", "\'", ":", "-", "[", "{", "]", "}", "=", "+", "`", "~", "/", "?", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "-" };
public readonly static string[] WikiInvalidChars = { "#", "<", ">", "[", "]", "|", "{", "}", "%", "+", "?", "\\", "/", ".", ",", ";", "-" };
public readonly static string[,] UrlReplacementStringValues =
{
		{ " = ", "Equals" },
		{ " == ", "Equals" },
		{ " != ", "NotEqual" },
		{ " \u2260 ", "NotEqual" },
		{ " > ", "Greater" },
		{ " \u2265 " , "GreaterOrEqual" },
		{ " >= ", "GreaterOrEqual" },
		{ " < ", "Less" },
		{ " \u2264 ", "LessOrEqual" },
		{ " <= ", "LessOrEqual" },
		{ " ", "_" },
		{ "[", string.Empty },
		{ "]", string.Empty }
};
public readonly static int UrlReplacementStringValuesLen = UrlReplacementStringValues.Length / 2;
public readonly static string[,] ReplacementStringValues =
{
		{ " = ", "Equals" },
		{ " == ", "Equals" },
		{ " != ", "NotEqual" },
		{ " \u2260 ", "NotEqual" },
		{ " > ", "Greater" },
		{ " \u2265 ", "GreaterOrEqual" },
		{ " >= ", "GreaterOrEqual" },
		{ " < ", "Less" },
		{ " \u2264 ", "LessOrEqual" },
		{ " <= ", "LessOrEqual" }
};
public readonly static int ReplacementStringValuesLen = ReplacementStringValues.Length / 2;
public readonly static string InternalData = "INTERNAL_DATA";
public readonly static string NoMaterialStr = "None";
public readonly static string OptionalParametersSep = " ";
public readonly static string NodeUndoId = "NODE_UNDO_ID";
public readonly static string NodeCreateUndoId = "NODE_CREATE_UNDO_ID";
public readonly static string NodeDestroyUndoId = "NODE_DESTROY_UNDO_ID";
public readonly static string CNIP = "#IP";
public readonly static float FLOAT_DRAW_HEIGHT_FIELD_SIZE = 16f;
public readonly static float FLOAT_DRAW_WIDTH_FIELD_SIZE = 45f;
public readonly static float FLOAT_WIDTH_SPACING = 3f;
public readonly static Color LockedPortColor = new Color( 0.3f, 0.3f, 0.3f, 0.5f );
public readonly static int[] AvailableUVChannels = { 0, 1, 2, 3, 4, 5, 6, 7 };
public readonly static string[] AvailableUVChannelsStr = { "0", "1", "2", "3", "4", "5", "6", "7"};
public readonly static string AvailableUVChannelLabel = "UV Channel";
public readonly static int[] AvailableUVSets = { 0, 1, 2, 3, 4, 5, 6, 7 };
public readonly static string[] AvailableUVSetsStr = { "1", "2", "3", "4","5", "6", "7", "8" };
public readonly static string AvailableUVSetsLabel = "UV Set";
public readonly static int[] AvailableUVChannels = { 0, 1, 2, 3 };
public readonly static string[] AvailableUVChannelsStr = { "0", "1", "2", "3" };
public readonly static string AvailableUVChannelLabel = "UV Channel";
public readonly static int[] AvailableUVSets = { 0, 1, 2, 3 };
public readonly static string[] AvailableUVSetsStr = { "1", "2", "3", "4" };
public readonly static string AvailableUVSetsLabel = "UV Set";
public readonly static int[] AvailableUVSizes = { 2, 3, 4 };
public readonly static string[] AvailableUVSizesStr = { "Float 2", "Float 3", "Float 4" };
public readonly static string AvailableUVSizesLabel = "Coord Size";
public readonly static string LineSeparator = "_____";
public readonly static Vector2 CopyPasteDeltaPos = new Vector2( 40, 40 );
public readonly static string[] VectorSuffixes = { ".x", ".y", ".z", ".w" };
public readonly static string[] ColorSuffixes = { ".r", ".g", ".b", ".a" };
public const string InternalDataLabelStr = "Internal Data";
public const string AttributesLaberStr = "Attributes";
public const string ParameterLabelStr = "Parameters";
public static readonly string[] ReferenceArrayLabels = { "Object", "Reference" };
public static readonly string[] ChannelNamesVector = { "X", "Y", "Z", "W" };
public static readonly string[] ChannelNamesColor = { "R", "G", "B", "A" };
public static readonly string SamplerFormat = "sampler{0}";
public static readonly string SamplerDeclFormat = "SamplerState {0}";
```

```
                public static readonly string SamplerDeclSRPFormat = "SAMPLER({0})";
        }
}
    Constants.cs.meta
    fileFormatVersion: 2
guid: d833dd0968f913f449477da6bcd56b48
timeCreated: 1481126959
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
    ASEBeginDecorator.cs
    using UnityEngine;
using UnityEditor;
using System;
using AmplifyShaderEditor;
public class ASEBeginDecorator : MaterialPropertyDrawer
{
        const int Separator = 2;
        public override void OnGUI( Rect position, MaterialProperty prop, String label, MaterialEditor editor )
        {
                Rect button = position;
                button.height = EditorGUIUtility.singleLineHeight;
                if( GUI.Button( button, "Open in Shader Editor" ) )
                {
                        Material mat = editor.target as Material;
                        ASEPackageManagerHelper.SetupLateMaterial( mat );
                        AmplifyShaderEditorWindow.LoadMaterialToASE( mat );
                }
        }
        public override float GetPropertyHeight( MaterialProperty prop, string label, MaterialEditor editor )
        {
                return EditorGUIUtility.singleLineHeight + Separator;
        }
}
    ASEBeginDecorator.cs.meta
    fileFormatVersion: 2
guid: 508788a7fa76e1d42ad5fdfb1c941ed2
MonoImporter:
  externalObjects: {}
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
    ASEEndDecorator.cs
    using UnityEngine;
using UnityEditor;
using System;
using AmplifyShaderEditor;
public class ASEEndDecorator : MaterialPropertyDrawer
{
        bool m_applyNext = false;
        public override void OnGUI( Rect position, MaterialProperty prop, String label, MaterialEditor editor )
        {
                if( prop.applyPropertyCallback == null )
                        prop.applyPropertyCallback = Testc;
                if( GUI.changed || m_applyNext )
                {
                        m_applyNext = false;
                        Material mat = editor.target as Material;
                        UIUtils.CopyValuesFromMaterial( mat );
                }
        }
        bool Testc( MaterialProperty prop, int changeMask, object previousValue )
        {
                m_applyNext = true;
                return false;
        }
        public override float GetPropertyHeight( MaterialProperty prop, string label, MaterialEditor editor )
        {
                return 0;
        }
}
    ASEEndDecorator.cs.meta
    fileFormatVersion: 2
guid: fdf2e52babbbbf040b3b9f6df50243f3
MonoImporter:
  externalObjects: {}
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
```

EditableIf.cs

```
using UnityEngine;
using UnityEditor;
using System;
public enum ComparisonOperators
{
	EqualTo, NotEqualTo, GreaterThan, LessThan, EqualsOrGreaterThan, EqualsOrLessThan, ContainsFlags,
	DoesNotContainsFlags
}
public class EditableIf : MaterialPropertyDrawer
{
	ComparisonOperators op;
	string FieldName = "";
	object ExpectedValue;
	bool InputError;
	public EditableIf()
	{
		InputError = true;
	}
	public EditableIf( object fieldname, object comparison, object expectedvalue )
	{
		if( expectedvalue.ToString().ToLower() == "true" )
		{
			expectedvalue = (System.Single)1;
		}
		else if( expectedvalue.ToString().ToLower() == "false" )
		{
			expectedvalue = (System.Single)0;
		}
		Init( fieldname, comparison, expectedvalue );
	}
	public EditableIf( object fieldname, object comparison, object expectedvaluex, object expectedvaluey )
	{
		float? x = expectedvaluex as float?;
		float? y = expectedvaluey as float?;
		float? z = float.NegativeInfinity;
		float? w = float.NegativeInfinity;
		x = GetVectorValue( x );
		y = GetVectorValue( y );
		Init( fieldname, comparison, new Vector4( x.Value, y.Value, z.Value, w.Value ) );
	}
	public EditableIf( object fieldname, object comparison, object expectedvaluex, object expectedvaluey, object expectedvaluez )
	{
		float? x = expectedvaluex as float?;
		float? y = expectedvaluey as float?;
		float? z = expectedvaluez as float?;
		float? w = float.NegativeInfinity;
		x = GetVectorValue( x );
		y = GetVectorValue( y );
		z = GetVectorValue( z );
		Init( fieldname, comparison, new Vector4( x.Value, y.Value, z.Value, w.Value ) );
	}
	public EditableIf( object fieldname, object comparison, object expectedvaluex, object expectedvaluey, object expectedvaluez, object expectedvaluew )
	{
		var x = expectedvaluex as float?;
		var y = expectedvaluey as float?;
		var z = expectedvaluez as float?;
		var w = expectedvaluew as float?;
		x = GetVectorValue( x );
		y = GetVectorValue( y );
		z = GetVectorValue( z );
		w = GetVectorValue( w );
		Init( fieldname, comparison, new Vector4( x.Value, y.Value, z.Value, w.Value ) );
	}
	private void Init( object fieldname, object comparison, object expectedvalue )
	{
		FieldName = fieldname.ToString();
		var names = Enum.GetNames( typeof( ComparisonOperators ) );
		var name = comparison.ToString().ToLower().Replace( " ", "" );
		for( int i = 0; i < names.Length; i++ )
		{
			if( names[ i ].ToLower() == name )
			{
				op = (ComparisonOperators)i;
				break;
			}
		}
		ExpectedValue = expectedvalue;
	}
	private static float? GetVectorValue( float? x )
	{
		if( x.HasValue == false )
		{
			x = float.NegativeInfinity;
		}
		return x;
	}
	public override void OnGUI( Rect position, MaterialProperty prop, String label, MaterialEditor editor )
	{
		if( InputError )
		{
			EditorGUI.LabelField( position, "EditableIf Attribute Error: Input parameters are invalid!" );
			return;
```

```
		}
		var LHSprop = MaterialEditor.GetMaterialProperty( prop.targets, FieldName );
		if( string.IsNullOrEmpty( LHSprop.name ) )
		{
			LHSprop = MaterialEditor.GetMaterialProperty( prop.targets, "_" + FieldName.Replace( " ", "" ) );
			if( string.IsNullOrEmpty( LHSprop.name ) )
			{
				EditorGUI.LabelField( position, "EditableIf Attribute Error: " + FieldName + " Does not exist!" );
				return;
			}
		}
		object LHSVal = null;
		bool test = false;
		switch( LHSprop.type )
		{
			case MaterialProperty.PropType.Color:
			case MaterialProperty.PropType.Vector:
			LHSVal = LHSprop.type == MaterialProperty.PropType.Color ? (Vector4)LHSprop.colorValue : LHSprop.vectorValue;
			var v4 = ExpectedValue as Vector4?;
			v4 = v4.HasValue ? v4 : new Vector4( (System.Single)ExpectedValue, float.NegativeInfinity, float.NegativeInfinity, float.NegativeInfinity );
			if( LHSprop.type == MaterialProperty.PropType.Color )
			{
				test = VectorCheck( (Vector4)LHSVal, op, v4 / 255 );
			}
			else
				test = VectorCheck( (Vector4)LHSVal, op, v4 );
			break;
			case MaterialProperty.PropType.Range:
			case MaterialProperty.PropType.Float:
			LHSVal = LHSprop.floatValue;
			test = ( Check( LHSVal, op, ExpectedValue ) );
			break;
			case MaterialProperty.PropType.Texture:
			LHSVal = LHSprop.textureValue;
			test = ( CheckObject( LHSVal, op, ExpectedValue ) );
			break;
		}
		GUI.enabled = test;
		editor.DefaultShaderProperty( position, prop, label );
		GUI.enabled = true;
	}
	private bool VectorCheck( Vector4 LHS, ComparisonOperators op, object expectedValue )
	{
		var RHS = (Vector4)expectedValue;
		if( RHS.x != float.NegativeInfinity )
		{
			if( !Check( LHS.x, op, RHS.x ) )
				return false;
		}
		if( RHS.y != float.NegativeInfinity )
		{
			if( !Check( LHS.y, op, RHS.y ) )
				return false;
		}
		if( RHS.z != float.NegativeInfinity )
		{
			if( !Check( LHS.z, op, RHS.z ) )
				return false;
		}
		if( RHS.w != float.NegativeInfinity )
		{
			if( !Check( LHS.w, op, RHS.w ) )
				return false;
		}
		return true;
	}
	protected bool Check( object LHS, ComparisonOperators op, object RHS )
	{
		if( !( LHS is IComparable ) || !( RHS is IComparable ) )
			throw new Exception( "Check using non basic type" );
		switch( op )
		{
			case ComparisonOperators.EqualTo:
			return ( (IComparable)LHS ).CompareTo( RHS ) == 0;
			case ComparisonOperators.NotEqualTo:
			return ( (IComparable)LHS ).CompareTo( RHS ) != 0;
			case ComparisonOperators.EqualsOrGreaterThan:
			return ( (IComparable)LHS ).CompareTo( RHS ) >= 0;
			case ComparisonOperators.EqualsOrLessThan:
			return ( (IComparable)LHS ).CompareTo( RHS ) <= 0;
			case ComparisonOperators.GreaterThan:
			return ( (IComparable)LHS ).CompareTo( RHS ) > 0;
			case ComparisonOperators.LessThan:
			return ( (IComparable)LHS ).CompareTo( RHS ) < 0;
			case ComparisonOperators.ContainsFlags:
			return ( (int)LHS & (int)RHS ) != 0; // Dont trust LHS values, it has been casted to a char and then to an int again, first bit will be the sign
			case ComparisonOperators.DoesNotContainsFlags:
			return ( ( (int)LHS & (int)RHS ) == (int)LHS ); // Dont trust LHS values, it has been casted to a char and then to an int again, first bit will be the sign
			default:
			break;
		}
		return false;
	}
```

```
		private bool CheckObject( object LHS, ComparisonOperators comparasonOperator, object RHS )
		{
			switch( comparasonOperator )
			{
				case ComparisonOperators.EqualTo:
				return ( LHS == null );
				case ComparisonOperators.NotEqualTo:
				return ( LHS != null );
			}
			return true;
		}
}
```

EditableIf.cs.meta

```
fileFormatVersion: 2
guid: 7a5504a2b7d04a846978416748dc6e0a
timeCreated: 1520330108
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
```

NoKeywordToggle.cs

```
using UnityEngine;
using UnityEditor;
using System;
public class NoKeywordToggle : MaterialPropertyDrawer
{
    public override void OnGUI(Rect position, MaterialProperty prop, String label, MaterialEditor editor) {
        bool value = (prop.floatValue != 0.0f);
        EditorGUI.BeginChangeCheck();
            {
                EditorGUI.showMixedValue = prop.hasMixedValue;
                value = EditorGUI.Toggle( position, label, value );
                EditorGUI.showMixedValue = false;
            }
        if (EditorGUI.EndChangeCheck())
            {
            prop.floatValue = value ? 1.0f : 0.0f;
        }
    }
}
```

NoKeywordToggle.cs.meta

```
fileFormatVersion: 2
guid: e1a000d43a26286499b39a7571e5c61b
timeCreated: 1605540234
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
```

RemapSliders.cs

```
using UnityEngine;
using UnityEditor;
using System;
public class RemapSliders : MaterialPropertyDrawer
{
	public override void OnGUI( Rect position, MaterialProperty prop, String label, MaterialEditor editor )
	{
		EditorGUI.BeginChangeCheck();
		Vector4 value = prop.vectorValue;
		EditorGUI.showMixedValue = prop.hasMixedValue;
		var cacheLabel = EditorGUIUtility.labelWidth;
		var cacheField = EditorGUIUtility.fieldWidth;
		if( cacheField <= 64 )
		{
			float total = position.width;
			EditorGUIUtility.labelWidth = Mathf.Ceil( 0.45f * total ) - 30;
			EditorGUIUtility.fieldWidth = Mathf.Ceil( 0.55f * total ) + 30;
		}
		EditorGUI.MinMaxSlider(position, label, ref value.x, ref value.y, 0, 1 );
		EditorGUIUtility.labelWidth = cacheLabel;
		EditorGUIUtility.fieldWidth = cacheField;
		EditorGUI.showMixedValue = false;
		if( EditorGUI.EndChangeCheck() )
		{
			prop.vectorValue = value;
		}
	}
}
```

RemapSliders.cs.meta

```
fileFormatVersion: 2
guid: 314af1bcecbba6c4d92cbb5843c221ba
MonoImporter:
  externalObjects: {}
  serializedVersion: 2
```

```
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
    RemapSlidersFull.cs
    using UnityEngine;
using UnityEditor;
using System;
public class RemapSlidersFull : MaterialPropertyDrawer
{
	public override void OnGUI( Rect position, MaterialProperty prop, String label, MaterialEditor editor )
	{
		EditorGUI.BeginChangeCheck();
		Vector4 value = prop.vectorValue;
		EditorGUI.showMixedValue = prop.hasMixedValue;
		var cacheLabel = EditorGUIUtility.labelWidth;
		var cacheField = EditorGUIUtility.fieldWidth;
		if( cacheField <= 64 )
		{
			float total = position.width;
			EditorGUIUtility.labelWidth = Mathf.Ceil( 0.45f * total ) - 30;
			EditorGUIUtility.fieldWidth = Mathf.Ceil( 0.55f * total ) + 30;
		}
		EditorGUI.MinMaxSlider( position, label, ref value.x, ref value.y, value.z, value.w );
		EditorGUIUtility.labelWidth = cacheLabel;
		EditorGUIUtility.fieldWidth = cacheField;
		EditorGUI.showMixedValue = false;
		if( EditorGUI.EndChangeCheck() )
		{
			prop.vectorValue = value;
		}
	}
}
    RemapSlidersFull.cs.meta
    fileFormatVersion: 2
guid: 9a724dcf5c5ddef40bcef06f0b2c8ec0
MonoImporter:
  externalObjects: {}
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
    SingleLineTexture.cs
    using UnityEngine;
using UnityEditor;
using System;
public class SingleLineTexture : MaterialPropertyDrawer
{
	public override void OnGUI( Rect position, MaterialProperty prop, String label, MaterialEditor editor )
	{
		EditorGUI.BeginChangeCheck();
		EditorGUI.showMixedValue = prop.hasMixedValue;
		Texture value = editor.TexturePropertyMiniThumbnail( position, prop, label, string.Empty );
		EditorGUI.showMixedValue = false;
		if( EditorGUI.EndChangeCheck() )
		{
			prop.textureValue = value;
		}
	}
}
    SingleLineTexture.cs.meta
    fileFormatVersion: 2
guid: 85da32683d237ac4f8665251e2ac38dc
MonoImporter:
  externalObjects: {}
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
    CustomDrawers.meta
    fileFormatVersion: 2
guid: 2206c4bd7f3d18643a6a3452b0c070d1
folderAsset: yes
timeCreated: 1522769470
licenseType: Store
DefaultImporter:
  userData:
  assetBundleName:
  assetBundleVariant:
    DoCreateFunction.cs
    using UnityEditor;
using UnityEditor.ProjectWindowCallback;
namespace AmplifyShaderEditor
{
	public class DoCreateFunction : EndNameEditAction
```

```
		{
			public override void Action( int instanceId, string pathName, string resourceFile )
			{
				UnityEngine.Object obj = EditorUtility.InstanceIDToObject( instanceId );
				AssetDatabase.CreateAsset( obj, AssetDatabase.GenerateUniqueAssetPath( pathName ) );
				AmplifyShaderEditorWindow.LoadShaderFunctionToASE( (AmplifyShaderFunction)obj, false );
			}
		}
}
	DoCreateFunction.cs.meta
	fileFormatVersion: 2
guid: 3f2c950b0ed192943b7484f6b551965f
timeCreated: 1493906087
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
	DoCreateShader.cs
	using UnityEngine;
using UnityEditor;
using UnityEditor.ProjectWindowCallback;
using System.IO;
namespace AmplifyShaderEditor
{
		public class DoCreateStandardShader : EndNameEditAction
		{
			public override void Action( int instanceId, string pathName, string resourceFile )
			{
				string uniquePath = AssetDatabase.GenerateUniqueAssetPath( pathName );
				string shaderName = Path.GetFileName( uniquePath );
				if( IOUtils.AllOpenedWindows.Count > 0 )
				{
					EditorWindow openedWindow = AmplifyShaderEditorWindow.GetWindow<AmplifyShaderEditorWindow>();
					AmplifyShaderEditorWindow currentWindow = AmplifyShaderEditorWindow.CreateTab();
					WindowHelper.AddTab( openedWindow, currentWindow );
					UIUtils.CurrentWindow = currentWindow;
				}
				else
				{
					AmplifyShaderEditorWindow currentWindow = AmplifyShaderEditorWindow.OpenWindow( shaderName, UIUtils.ShaderIcon );
					UIUtils.CurrentWindow = currentWindow;
				}
				Shader shader = UIUtils.CreateNewEmpty( uniquePath, shaderName );
				ProjectWindowUtil.ShowCreatedAsset( shader );
			}
		}
		public class DoCreateTemplateShader : EndNameEditAction
		{
			public override void Action( int instanceId, string pathName, string resourceFile )
			{
				string uniquePath = AssetDatabase.GenerateUniqueAssetPath( pathName );
				string shaderName = Path.GetFileName( uniquePath );
				if( !string.IsNullOrEmpty( UIUtils.NewTemplateGUID ) )
				{
					Shader shader = AmplifyShaderEditorWindow.CreateNewTemplateShader( UIUtils.NewTemplateGUID, uniquePath, shaderName );
					ProjectWindowUtil.ShowCreatedAsset( shader );
				}
			}
		}
}
	DoCreateShader.cs.meta
	fileFormatVersion: 2
guid: 2cfa7290f61ad684f99f8d81328ad52c
timeCreated: 1573664425
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
	EditorOptions.cs
	using UnityEditor;
namespace AmplifyShaderEditor
{
		[System.Serializable]
		public class OptionsWindow
		{
			private AmplifyShaderEditorWindow m_parentWindow = null;
			private bool m_coloredPorts = true;
			private bool m_multiLinePorts = true;
			private const string MultiLineId = "MultiLinePortsDefault";
			private const string ColorPortId = "ColoredPortsDefault";
			public OptionsWindow( AmplifyShaderEditorWindow parentWindow )
			{
				m_parentWindow = parentWindow;
```

```
		}
		public void Init()
		{
			Load();
		}
		public void Destroy()
		{
			Save();
		}
		public void Save()
		{
			EditorPrefs.SetBool( ColorPortId, ColoredPorts );
			EditorPrefs.SetBool( MultiLineId, m_multiLinePorts );
		}
		public void Load()
		{
			ColoredPorts = EditorPrefs.GetBool( ColorPortId, true );
			m_multiLinePorts = EditorPrefs.GetBool( MultiLineId, true );
		}
		public bool ColoredPorts
		{
			get { return m_coloredPorts; }
			set
			{
				if ( m_coloredPorts != value )
					EditorPrefs.SetBool( ColorPortId, value );
				m_coloredPorts = value;
			}
		}
		public bool MultiLinePorts
		{
			get { return m_multiLinePorts; }
			set
			{
				if ( m_multiLinePorts != value )
					EditorPrefs.SetBool( MultiLineId, value );
				m_multiLinePorts = value;
			}
		}
		public AmplifyShaderEditorWindow ParentWindow { get { return m_parentWindow; } set { m_parentWindow = value; } }
	}
}
	EditorOptions.cs.meta
	fileFormatVersion: 2
guid: 44cb06bc7bfe6e84aa8b5e8b702eb2dd
timeCreated: 1481126955
licenseType: Store
MonoImporter:
  serializedVersion: 2
  defaultReferences: []
  executionOrder: 0
  icon: {instanceID: 0}
  userData:
  assetBundleName:
  assetBundleVariant:
	NodeGrid.cs
	using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
namespace AmplifyShaderEditor
{
	public class NodeGrid
	{
		private bool m_debugGrid = false;
		private const float GRID_SIZE_X = 100;
		private const float GRID_SIZE_Y = 100;
		private const float GRID_AREA_X = 1000;
		private const float GRID_AREA_Y = 1000;
		private Dictionary<int, Dictionary<int, List<ParentNode>>> m_grid;
		private int m_xMin = int.MaxValue;
		private int m_yMin = int.MaxValue;
		private int m_xMax = int.MinValue;
		private int m_yMax = int.MinValue;
		public NodeGrid()
		{
			m_grid = new Dictionary<int, Dictionary<int, List<ParentNode>>>();
		}
		public void AddNodeToGrid( ParentNode node )
		{
			Rect pos = node.Position;
			if ( Mathf.Abs( pos.width ) < 0.001f || Mathf.Abs( pos.height ) < 0.001f )
			{
				return;
			}
			float initialXf = pos.x / GRID_SIZE_X;
			float initialYf = pos.y / GRID_SIZE_Y;
			int endX = Mathf.CeilToInt( initialXf + pos.width / GRID_SIZE_X );
			int endY = Mathf.CeilToInt( initialYf + pos.height / GRID_SIZE_Y );
			int initialX = Mathf.FloorToInt( initialXf );
			int initialY = Mathf.FloorToInt( initialYf );
			if ( initialX < m_xMin )
			{
				m_xMin = initialX;
```

```
		}
		if ( initialY < m_yMin )
		{
			m_yMin = initialY;
		}
		if ( endX > m_xMax )
		{
			m_xMax = endX;
		}
		if ( endY > m_yMax )
		{
			m_yMax = endY;
		}
		for ( int x = initialX; x < endX; x += 1 )
		{
			for ( int y = initialY; y < endY; y += 1 )
			{
				if ( !m_grid.ContainsKey( x ) )
				{
					m_grid.Add( x, new Dictionary<int, List<ParentNode>>() );
				}
				if ( !m_grid[ x ].ContainsKey( y ) )
				{
					m_grid[ x ].Add( y, new List<ParentNode>() );
				}
				m_grid[ x ][ y ].Add( node );
			}
		}
		node.IsOnGrid = true;
	}
	public void RemoveNodeFromGrid( ParentNode node, bool useCachedPos )
	{
		Rect pos = useCachedPos ? node.CachedPos : node.Position;
		if ( Mathf.Abs( pos.width ) < 0.001f || Mathf.Abs( pos.height ) < 0.001f )
		{
			return;
		}
		float initialXf = pos.x / GRID_SIZE_X;
		float initialYf = pos.y / GRID_SIZE_Y;
		int endX = Mathf.CeilToInt( initialXf + pos.width / GRID_SIZE_X );
		int endY = Mathf.CeilToInt( initialYf + pos.height / GRID_SIZE_Y );
		int initialX = Mathf.FloorToInt( initialXf );
		int initialY = Mathf.FloorToInt( initialYf );
		bool testLimits = false;
		int xMinCount = 0;
		int xMaxCount = 0;
		int yMinCount = 0;
		int yMaxCount = 0;
		for ( int x = initialX; x < endX; x += 1 )
		{
			for ( int y = initialY; y < endY; y += 1 )
			{
				if ( m_grid.ContainsKey( x ) )
				{
					if ( m_grid[ x ].ContainsKey( y ) )
					{
						m_grid[ x ][ y ].Remove( node );
						node.IsOnGrid = false;
						if ( initialX == m_xMin && x == initialX )
						{
							testLimits = true;
							if ( m_grid[ x ][ y ].Count != 0 )
							{
								xMinCount += 1;
							}
						}
						if ( endX == m_xMax && x == endX )
						{
							testLimits = true;
							if ( m_grid[ x ][ y ].Count != 0 )
							{
								xMaxCount += 1;
							}
						}
						if ( initialY == m_yMin && y == initialY )
						{
							testLimits = true;
							if ( m_grid[ x ][ y ].Count != 0 )
							{
								yMinCount += 1;
							}
						}
						if ( endY == m_yMax && y == endY )
						{
							testLimits = true;
							if ( m_grid[ x ][ y ].Count != 0 )
							{
								yMaxCount += 1;
							}
						}
					}
				}
			}
```

```
                }
                if ( testLimits )
                {
                        if ( xMinCount == 0 || xMaxCount == 0 || yMinCount == 0 || yMaxCount == 0 )
                        {
                                m_xMin = int.MaxValue;
                                m_yMin = int.MaxValue;
                                m_xMax = int.MinValue;
                                m_yMax = int.MinValue;
                                foreach ( KeyValuePair<int, Dictionary<int, List<ParentNode>>> entryX in m_grid )
                                {
                                        foreach ( KeyValuePair<int, List<ParentNode>> entryY in entryX.Value )
                                        {
                                                if ( entryY.Value.Count > 0 )
                                                {
                                                        if ( entryX.Key < m_xMin )
                                                        {
                                                                m_xMin = entryX.Key;
                                                        }
                                                        if ( entryY.Key < m_yMin )
                                                        {
                                                                m_yMin = entryY.Key;
                                                        }
                                                        if ( entryX.Key > m_xMax )
                                                        {
                                                                m_xMax = entryX.Key;
                                                        }
                                                        if ( entryY.Key > m_yMax )
                                                        {
                                                                m_yMax = entryY.Key;
                                                        }
                                                }
                                        }
                                }
                                m_xMax += 1;
                                m_yMax += 1;
                        }
                }
        }
        public void DebugLimits()
        {
                Debug.Log( "[ " + m_xMin + " , " + m_yMin + " ] " + "[ " + m_xMax + " , " + m_yMax + " ] " );
        }
        public List<ParentNode> GetNodesOn( Vector2 pos )
        {
                int x = Mathf.FloorToInt( pos.x / GRID_SIZE_X );
                int y = Mathf.FloorToInt( pos.y / GRID_SIZE_Y );
                if ( m_grid.ContainsKey( x ) )
                {
                        if ( m_grid[ x ].ContainsKey( y ) )
                        {
                                return m_grid[ x ][ y ];
                        }
                }
                return null;
        }
        public List<ParentNode> GetNodesOn( int x, int y )
        {
                if ( m_grid.ContainsKey( x ) )
                {
                        if ( m_grid[ x ].ContainsKey( y ) )
                        {
                                return m_grid[ x ][ y ];
                        }
                }
                return null;
        }
        public void DrawGrid( DrawInfo drawInfo )
        {
                if ( m_debugGrid )
                {
                        Handles.CircleHandleCap( 0, drawInfo.InvertedZoom * ( new Vector3( drawInfo.CameraOffset.x, drawInfo.CameraOffset.y, 0f ) ), Quaternion.identity, 5,EventType.Layout );
                        for ( int x = -( int ) GRID_AREA_X; x < GRID_AREA_X; x += ( int ) GRID_SIZE_X )
                        {
                                Handles.DrawLine( drawInfo.InvertedZoom * ( new Vector3( x + drawInfo.CameraOffset.x, drawInfo.CameraOffset.y - GRID_AREA_Y, 0 ) ), drawInfo.InvertedZoom * ( new Vector3( drawInfo.CameraOffset.x + x, drawInfo.CameraOffset.y + GRID_AREA_Y, 0 ) ) );
                        }
                        for ( int y = -( int ) GRID_AREA_Y; y < GRID_AREA_X; y += ( int ) GRID_SIZE_Y )
                        {
                                Handles.DrawLine( drawInfo.InvertedZoom * ( new Vector3( drawInfo.CameraOffset.x - GRID_AREA_X, drawInfo.CameraOffset.y + y, 0 ) ), drawInfo.InvertedZoom * ( new Vector3( drawInfo.CameraOffset.x + GRID_AREA_X, drawInfo.CameraOffset.y + y, 0 ) ) );
                        }
                }
        }
        public void Destroy()
        {
                foreach ( KeyValuePair<int, Dictionary<int, List<ParentNode>>> entryX in m_grid )
                {
                        foreach ( KeyValuePair<int, List<ParentNode>> entryY in entryX.Value )
                        {
                                entryY.Value.Clear();
                        }
                        entryX.Value.Clear();
                }
                m_grid.Clear();
```

```
                }
                public float MaxNodeDist
                {
                        get { return Mathf.Max( ( m_xMax - m_xMin )*GRID_SIZE_X, ( m_yMax - m_yMin )*GRID_SIZE_Y ); }
                }
        }
}
        NodeGrid.cs.meta
        fileFormatVersion: 2
guid: 6344917ce0eed6b43840632b98a2ed57
timeCreated: 1481126956
licenseType: Store
MonoImporter:
    serializedVersion: 2
    defaultReferences: []
    executionOrder: 0
    icon: {instanceID: 0}
    userData:
    assetBundleName:
    assetBundleVariant:
        ParentGraph.cs
        using UnityEngine;
using UnityEditor;
using System;
using System.Collections.Generic;
namespace AmplifyShaderEditor
{
        [Serializable]
        public class ParentGraph : ScriptableObject, ISerializationCallbackReceiver
        {
                private const int MasterNodeLODIncrement = 100;
                private const int MaxLodAmount = 9;
                public enum NodeLOD
                {
                        LOD0,
                        LOD1,
                        LOD2,
                        LOD3,
                        LOD4,
                        LOD5
                }
                [SerializeField]
                private bool m_samplingThroughMacros = false;
                private NodeLOD m_lodLevel = NodeLOD.LOD0;
                private GUIStyle nodeStyleOff;
                private GUIStyle nodeStyleOn;
                private GUIStyle nodeTitle;
                private GUIStyle commentaryBackground;
                public delegate void LODMasterNodesAdded( int lod );
                public event LODMasterNodesAdded OnLODMasterNodesAddedEvent;
                public delegate void EmptyGraphDetected( ParentGraph graph );
                public event EmptyGraphDetected OnEmptyGraphDetectedEvt;
                public delegate void NodeEvent( ParentNode node );
                public event NodeEvent OnNodeEvent = null;
                public event NodeEvent OnNodeRemovedEvent;
                public delegate void DuplicateEvent();
                public event DuplicateEvent OnDuplicateEvent;
                public event MasterNode.OnMaterialUpdated OnMaterialUpdatedEvent;
                public event MasterNode.OnMaterialUpdated OnShaderUpdatedEvent;
                private bool m_afterDeserializeFlag = true;
                private bool m_lateOptionsRefresh = false;
                private bool m_foundDuplicates = false;
                private AmplifyShaderEditorWindow m_parentWindow = null;
                [SerializeField]
                private int m_validNodeId;
                [SerializeField]
                private List<ParentNode> m_nodes = new List<ParentNode>();
                [SerializeField]
                private UsageListSamplerNodes m_samplerNodes = new UsageListSamplerNodes();
                [SerializeField]
                private UsageListFloatIntNodes m_floatNodes = new UsageListFloatIntNodes();
                [SerializeField]
                private UsageListTexturePropertyNodes m_texturePropertyNodes = new UsageListTexturePropertyNodes();
                [SerializeField]
                private UsageListTextureArrayNodes m_textureArrayNodes = new UsageListTextureArrayNodes();
                [SerializeField]
                private UsageListPropertyNodes m_propertyNodes = new UsageListPropertyNodes();
                [SerializeField]
                private UsageListPropertyNodes m_rawPropertyNodes = new UsageListPropertyNodes();
                [SerializeField]
                private UsageListScreenColorNodes m_screenColorNodes = new UsageListScreenColorNodes();
                [SerializeField]
                private UsageListRegisterLocalVarNodes m_localVarNodes = new UsageListRegisterLocalVarNodes();
                [SerializeField]
                private UsageListGlobalArrayNodes m_globalArrayNodes = new UsageListGlobalArrayNodes();
                [SerializeField]
                private UsageListFunctionInputNodes m_functionInputNodes = new UsageListFunctionInputNodes();
                [SerializeField]
                private UsageListFunctionNodes m_functionNodes = new UsageListFunctionNodes();
                [SerializeField]
                private UsageListFunctionOutputNodes m_functionOutputNodes = new UsageListFunctionOutputNodes();
                [SerializeField]
                private UsageListFunctionSwitchNodes m_functionSwitchNodes = new UsageListFunctionSwitchNodes();
```

```
[SerializeField]
private UsageListFunctionSwitchCopyNodes m_functionSwitchCopyNodes = new UsageListFunctionSwitchCopyNodes();
[SerializeField]
private UsageListTemplateMultiPassMasterNodes m_multiPassMasterNodes = new UsageListTemplateMultiPassMasterNodes();
[SerializeField]
private List<UsageListTemplateMultiPassMasterNodes> m_lodMultiPassMasterNodes;
[SerializeField]
private UsageListCustomExpressionsOnFunctionMode m_customExpressionsOnFunctionMode = new UsageListCustomExpressionsOnFunctionMode();
[SerializeField]
private UsageListStaticSwitchNodes m_staticSwitchNodes = new UsageListStaticSwitchNodes();
[SerializeField]
private int m_masterNodeId = Constants.INVALID_NODE_ID;
[SerializeField]
private bool m_isDirty;
[SerializeField]
private bool m_saveIsDirty = false;
[SerializeField]
private int m_nodeClicked;
[SerializeField]
private int m_loadedShaderVersion;
[SerializeField]
private int m_instancePropertyCount = 0;
[SerializeField]
private int m_virtualTextureCount = 0;
[SerializeField]
private int m_graphId = 0;
[SerializeField]
private PrecisionType m_currentPrecision = PrecisionType.Float;
[SerializeField]
private NodeAvailability m_currentCanvasMode = NodeAvailability.SurfaceShader;
[SerializeField]
private TemplateSRPType m_currentSRPType = TemplateSRPType.BuiltIn;
private List<ParentNode> m_nodePreviewList = new List<ParentNode>();
private Dictionary<int, ParentNode> m_nodesDict = new Dictionary<int, ParentNode>();
[NonSerialized]
private List<ParentNode> m_selectedNodes = new List<ParentNode>();
[NonSerialized]
private List<ParentNode> m_markedForDeletion = new List<ParentNode>();
[SerializeField]
private List<WireReference> m_highlightedWires = new List<WireReference>();
private System.Type m_masterNodeDefaultType;
[SerializeField]
private List<PropertyNode> m_internalTemplateNodesList = new List<PropertyNode>();
private Dictionary<int, PropertyNode> m_internalTemplateNodesDict = new Dictionary<int, PropertyNode>();
private NodeGrid m_nodeGrid;
private bool m_markedToDeSelect = false;
private int m_markToSelect = -1;
private bool m_markToReOrder = false;
private bool m_hasUnConnectedNodes = false;
private bool m_checkSelectedWireHighlights = false;
[SerializeField]
private List<WireBezierReference> m_bezierReferences;
private const int MaxBezierReferences = 50;
private int m_wireBezierCount = 0;
protected int m_normalDependentCount = 0;
private bool m_forceCategoryRefresh = false;
[SerializeField]
private bool m_forceRepositionCheck = false;
private bool m_isLoading = false;
private bool m_isDuplicating = false;
private bool m_changedLightingModel = false;
public void ResetEvents()
{
	OnNodeEvent = null;
	OnMaterialUpdatedEvent = null;
	OnShaderUpdatedEvent = null;
	OnEmptyGraphDetectedEvt = null;
	OnNodeRemovedEvent = null;
}
public void Init()
{
	Undo.undoRedoPerformed += OnUndoRedoCallback;
	m_normalDependentCount = 0;
	m_nodes = new List<ParentNode>();
	m_samplerNodes = new UsageListSamplerNodes();
	m_samplerNodes.ContainerGraph = this;
	m_samplerNodes.ReorderOnChange = true;
	m_floatNodes = new UsageListFloatIntNodes();
	m_floatNodes.ContainerGraph = this;
	m_texturePropertyNodes = new UsageListTexturePropertyNodes();
	m_texturePropertyNodes.ContainerGraph = this;
	m_textureArrayNodes = new UsageListTextureArrayNodes();
	m_textureArrayNodes.ContainerGraph = this;
	m_textureArrayNodes.ReorderOnChange = true;
	m_propertyNodes = new UsageListPropertyNodes();
	m_propertyNodes.ContainerGraph = this;
	m_rawPropertyNodes = new UsageListPropertyNodes();
	m_rawPropertyNodes.ContainerGraph = this;
	m_customExpressionsOnFunctionMode = new UsageListCustomExpressionsOnFunctionMode();
	m_customExpressionsOnFunctionMode.ContainerGraph = this;
	m_staticSwitchNodes = new UsageListStaticSwitchNodes();
	m_staticSwitchNodes.ContainerGraph = this;
	m_staticSwitchNodes.ReorderOnChange = true;
```

```
		m_screenColorNodes = new UsageListScreenColorNodes();
		m_screenColorNodes.ContainerGraph = this;
		m_screenColorNodes.ReorderOnChange = true;
		m_localVarNodes = new UsageListRegisterLocalVarNodes();
		m_localVarNodes.ContainerGraph = this;
		m_localVarNodes.ReorderOnChange = true;
		m_globalArrayNodes = new UsageListGlobalArrayNodes();
		m_globalArrayNodes.ContainerGraph = this;
		m_functionInputNodes = new UsageListFunctionInputNodes();
		m_functionInputNodes.ContainerGraph = this;
		m_functionNodes = new UsageListFunctionNodes();
		m_functionNodes.ContainerGraph = this;
		m_functionOutputNodes = new UsageListFunctionOutputNodes();
		m_functionOutputNodes.ContainerGraph = this;
		m_functionSwitchNodes = new UsageListFunctionSwitchNodes();
		m_functionSwitchNodes.ContainerGraph = this;
		m_functionSwitchCopyNodes = new UsageListFunctionSwitchCopyNodes();
		m_functionSwitchCopyNodes.ContainerGraph = this;
		m_multiPassMasterNodes = new UsageListTemplateMultiPassMasterNodes();
		m_multiPassMasterNodes.ContainerGraph = this;
		m_lodMultiPassMasterNodes = new List<UsageListTemplateMultiPassMasterNodes>( MaxLodAmount );
		for( int i = 0; i < MaxLodAmount; i++ )
		{
			m_lodMultiPassMasterNodes.Add( new UsageListTemplateMultiPassMasterNodes() );
		}
		m_selectedNodes = new List<ParentNode>();
		m_markedForDeletion = new List<ParentNode>();
		m_highlightedWires = new List<WireReference>();
		m_validNodeId = 0;
		IsDirty = false;
		SaveIsDirty = false;
		m_masterNodeDefaultType = typeof( StandardSurfaceOutputNode );
		m_bezierReferences = new List<WireBezierReference>( MaxBezierReferences );
		for( int i = 0; i < MaxBezierReferences; i++ )
		{
			m_bezierReferences.Add( new WireBezierReference() );
		}
	}
	public void ActivatePreviews( bool value )
	{
		int count = m_nodes.Count;
		if( value )
		{
			for( int i = 0 ; i < count ; i++ )
			{
				m_nodes[ i ].PreviewIsDirty = true;
			}
		}
		else
		{
		}
	}
	private void OnUndoRedoCallback()
	{
		DeSelectAll();
	}
	private void OnEnable()
	{
		hideFlags = HideFlags.HideAndDontSave;
		m_nodeGrid = new NodeGrid();
		m_internalTemplateNodesDict = new Dictionary<int, PropertyNode>();
		m_nodesDict = new Dictionary<int, ParentNode>();
		nodeStyleOff = UIUtils.GetCustomStyle( CustomStyle.NodeWindowOff );
		nodeStyleOn = UIUtils.GetCustomStyle( CustomStyle.NodeWindowOn );
		nodeTitle = UIUtils.GetCustomStyle( CustomStyle.NodeHeader );
		commentaryBackground = UIUtils.GetCustomStyle( CustomStyle.CommentaryBackground );
	}
	public void UpdateRegisters()
	{
		m_samplerNodes.UpdateNodeArr();
		m_propertyNodes.UpdateNodeArr();
		m_rawPropertyNodes.UpdateNodeArr();
		m_customExpressionsOnFunctionMode.UpdateNodeArr();
		m_staticSwitchNodes.UpdateNodeArr();
		m_functionInputNodes.UpdateNodeArr();
		m_functionNodes.UpdateNodeArr();
		m_functionOutputNodes.UpdateNodeArr();
		m_functionSwitchNodes.UpdateNodeArr();
		m_functionSwitchCopyNodes.UpdateNodeArr();
		m_multiPassMasterNodes.UpdateNodeArr();
		for( int i = 0; i < m_lodMultiPassMasterNodes.Count; i++ )
		{
			m_lodMultiPassMasterNodes[ i ].UpdateNodeArr();
		}
		m_texturePropertyNodes.UpdateNodeArr();
		m_textureArrayNodes.UpdateNodeArr();
		m_screenColorNodes.UpdateNodeArr();
		m_localVarNodes.UpdateNodeArr();
		m_globalArrayNodes.UpdateNodeArr();
	}
	public int GetValidId()
	{
		return m_validNodeId++;
```

```
	}
	void UpdateIdFromNode( ParentNode node )
	{
		if( node.UniqueId >= m_validNodeId )
		{
			m_validNodeId = node.UniqueId + 1;
		}
	}
	public void ResetNodeConnStatus()
	{
		for( int i = 0; i < m_nodes.Count; i++ )
		{
			if( m_nodes[ i ].ConnStatus == NodeConnectionStatus.Connected )
			{
				m_nodes[ i ].ConnStatus = NodeConnectionStatus.Not_Connected;
			}
		}
	}
	public void CleanUnusedNodes()
	{
		List<ParentNode> unusedNodes = new List<ParentNode>();
		for( int i = 0; i < m_nodes.Count; i++ )
		{
			if( m_nodes[ i ].ConnStatus == NodeConnectionStatus.Not_Connected )
			{
				unusedNodes.Add( m_nodes[ i ] );
			}
		}
		for( int i = 0; i < unusedNodes.Count; i++ )
		{
			DestroyNode( unusedNodes[ i ] );
		}
		unusedNodes.Clear();
		unusedNodes = null;
		IsDirty = true;
	}
	public void ClearGraph()
	{
		List<ParentNode> list = new List<ParentNode>();
		int count = m_nodes.Count;
		for( int i = 0; i < count; i++ )
		{
			if( m_nodes[ i ].UniqueId != m_masterNodeId )
			{
				list.Add( m_nodes[ i ] );
			}
		}
		while( list.Count > 0 )
		{
			DestroyNode( list[ 0 ] );
			list.RemoveAt( 0 );
		}
	}
	public void CleanNodes()
	{
		for( int i = 0; i < m_nodes.Count; i++ )
		{
			if( m_nodes[ i ] != null )
			{
				Undo.ClearUndo( m_nodes[ i ] );
				m_nodes[ i ].Destroy();
				GameObject.DestroyImmediate( m_nodes[ i ] );
			}
		}
		ClearInternalTemplateNodes();
		m_masterNodeId = Constants.INVALID_NODE_ID;
		m_validNodeId = 0;
		m_instancePropertyCount = 0;
		m_virtualTextureCount = 0;
		m_nodesDict.Clear();
		m_nodes.Clear();
		m_samplerNodes.Clear();
		m_propertyNodes.Clear();
		m_rawPropertyNodes.Clear();
		m_customExpressionsOnFunctionMode.Clear();
		m_staticSwitchNodes.Clear();
		m_functionInputNodes.Clear();
		m_functionNodes.Clear();
		m_functionOutputNodes.Clear();
		m_functionSwitchNodes.Clear();
		m_functionSwitchCopyNodes.Clear();
		m_multiPassMasterNodes.Clear();
		for( int i = 0; i < m_lodMultiPassMasterNodes.Count; i++ )
		{
			m_lodMultiPassMasterNodes[ i ].Clear();
		}
		m_texturePropertyNodes.Clear();
		m_textureArrayNodes.Clear();
		m_screenColorNodes.Clear();
		m_localVarNodes.Clear();
		m_globalArrayNodes.Clear();
		m_selectedNodes.Clear();
		m_markedForDeletion.Clear();
```

```
	}
	public void ResetHighlightedWires()
	{
		for( int i = 0; i < m_highlightedWires.Count; i++ )
		{
			m_highlightedWires[ i ].WireStatus = WireStatus.Default;
		}
		m_highlightedWires.Clear();
	}
	public void HighlightWiresStartingNode( ParentNode node )
	{
		for( int outputIdx = 0; outputIdx < node.OutputPorts.Count; outputIdx++ )
		{
			for( int extIdx = 0; extIdx < node.OutputPorts[ outputIdx ].ExternalReferences.Count; extIdx++ )
			{
				WireReference wireRef = node.OutputPorts[ outputIdx ].ExternalReferences[ extIdx ];
				ParentNode nextNode = GetNode( wireRef.NodeId );
				if( nextNode && nextNode.ConnStatus == NodeConnectionStatus.Connected )
				{
					InputPort port = nextNode.GetInputPortByUniqueId( wireRef.PortId );
					if( port.ExternalReferences.Count == 0 || port.ExternalReferences[ 0 ].WireStatus == WireStatus.Highlighted )
					{
						return;
					}
					port.ExternalReferences[ 0 ].WireStatus = WireStatus.Highlighted;
					m_highlightedWires.Add( port.ExternalReferences[ 0 ] );
					HighlightWiresStartingNode( nextNode );
				}
			}
		}
		RegisterLocalVarNode regNode = node as RegisterLocalVarNode;
		if( (object)regNode != null )
		{
			int count = regNode.NodeReferences.Count;
			for( int i = 0; i < count; i++ )
			{
				HighlightWiresStartingNode( regNode.NodeReferences[ i ] );
			}
		}
	}
	void PropagateHighlightDeselection( ParentNode node, int portId = -1 )
	{
		if( portId > -1 )
		{
			InputPort port = node.GetInputPortByUniqueId( portId );
			port.ExternalReferences[ 0 ].WireStatus = WireStatus.Default;
		}
		if( node.Selected )
			return;
		for( int i = 0; i < node.InputPorts.Count; i++ )
		{
			if( node.InputPorts[ i ].ExternalReferences.Count > 0 && node.InputPorts[ i ].ExternalReferences[ 0 ].WireStatus == WireStatus.Highlighted )
			{
				return;
			}
		}
		for( int outputIdx = 0; outputIdx < node.OutputPorts.Count; outputIdx++ )
		{
			for( int extIdx = 0; extIdx < node.OutputPorts[ outputIdx ].ExternalReferences.Count; extIdx++ )
			{
				WireReference wireRef = node.OutputPorts[ outputIdx ].ExternalReferences[ extIdx ];
				ParentNode nextNode = GetNode( wireRef.NodeId );
				PropagateHighlightDeselection( nextNode, wireRef.PortId );
			}
		}
		RegisterLocalVarNode regNode = node as RegisterLocalVarNode;
		if( (object)regNode != null )
		{
			int count = regNode.NodeReferences.Count;
			for( int i = 0; i < count; i++ )
			{
				PropagateHighlightDeselection( regNode.NodeReferences[ i ], -1 );
			}
		}
	}
	public void ResetNodesData()
	{
		int count = m_nodes.Count;
		for( int i = 0; i < count; i++ )
		{
			m_nodes[ i ].ResetNodeData();
		}
	}
	public void FullCleanUndoStack()
	{
		Undo.ClearUndo( this );
		int count = m_nodes.Count;
		for( int i = 0; i < count; i++ )
		{
			if( m_nodes[ i ] != null )
			{
				Undo.ClearUndo( m_nodes[ i ] );
			}
```

```
		}
	}
	public void FullRegisterOnUndoStack()
	{
		Undo.RegisterCompleteObjectUndo( this, Constants.UndoRegisterFullGrapId );
		int count = m_nodes.Count;
		for( int i = 0; i < count; i++ )
		{
			if( m_nodes[ i ] != null )
			{
				Undo.RegisterCompleteObjectUndo( m_nodes[ i ], Constants.UndoRegisterFullGrapId );
			}
		}
	}
	public void CheckPropertiesAutoRegister( ref MasterNodeDataCollector dataCollector )
	{
		List<PropertyNode> propertyNodesList = m_rawPropertyNodes.NodesList;
		int propertyCount = propertyNodesList.Count;
		for( int i = 0; i < propertyCount; i++ )
		{
			propertyNodesList[ i ].CheckIfAutoRegister( ref dataCollector );
		}
		propertyNodesList = null;
		List<GlobalArrayNode> globalArrayNodeList = m_globalArrayNodes.NodesList;
		int globalArrayCount = globalArrayNodeList.Count;
		for( int i = 0; i < globalArrayCount; i++ )
		{
			globalArrayNodeList[ i ].CheckIfAutoRegister( ref dataCollector );
		}
		globalArrayNodeList = null;
	}
	public void SoftDestroy()
	{
		OnNodeRemovedEvent = null;
		m_masterNodeId = Constants.INVALID_NODE_ID;
		m_validNodeId = 0;
		m_nodeGrid.Destroy();
		ClearInternalTemplateNodes();
		for( int i = 0; i < m_nodes.Count; i++ )
		{
			if( m_nodes[ i ] != null )
			{
				m_nodes[ i ].Destroy();
				GameObject.DestroyImmediate( m_nodes[ i ] );
			}
		}
		m_instancePropertyCount = 0;
		m_nodes.Clear();
		m_nodesDict.Clear();
		m_samplerNodes.Clear();
		m_propertyNodes.Clear();
		m_rawPropertyNodes.Clear();
		m_customExpressionsOnFunctionMode.Clear();
		m_staticSwitchNodes.Clear();
		m_functionInputNodes.Clear();
		m_functionNodes.Clear();
		m_functionOutputNodes.Clear();
		m_functionSwitchNodes.Clear();
		m_functionSwitchCopyNodes.Clear();
		m_texturePropertyNodes.Clear();
		m_textureArrayNodes.Clear();
		m_screenColorNodes.Clear();
		m_localVarNodes.Clear();
		m_globalArrayNodes.Clear();
		m_selectedNodes.Clear();
		m_markedForDeletion.Clear();
		m_nodePreviewList.Clear();
		IsDirty = true;
		OnNodeEvent = null;
		OnDuplicateEvent = null;
		OnMaterialUpdatedEvent = null;
		OnShaderUpdatedEvent = null;
		OnEmptyGraphDetectedEvt = null;
		nodeStyleOff = null;
		nodeStyleOn = null;
		nodeTitle = null;
		commentaryBackground = null;
		OnLODMasterNodesAddedEvent = null;
	}
	public void Destroy()
	{
		Undo.undoRedoPerformed -= OnUndoRedoCallback;
		for( int i = 0; i < m_nodes.Count; i++ )
		{
			if( m_nodes[ i ] != null )
			{
				Undo.ClearUndo( m_nodes[ i ] );
				m_nodes[ i ].Destroy();
				GameObject.DestroyImmediate( m_nodes[ i ] );
			}
		}
		ClearInternalTemplateNodes();
		m_internalTemplateNodesDict = null;
```

```
		m_internalTemplateNodesList = null;
		OnNodeRemovedEvent = null;
		m_masterNodeId = Constants.INVALID_NODE_ID;
		m_validNodeId = 0;
		m_instancePropertyCount = 0;
		m_nodeGrid.Destroy();
		m_nodeGrid = null;
		m_nodes.Clear();
		m_nodes = null;
		m_samplerNodes.Destroy();
		m_samplerNodes = null;
		m_propertyNodes.Destroy();
		m_propertyNodes = null;
		m_rawPropertyNodes.Destroy();
		m_rawPropertyNodes = null;
		m_customExpressionsOnFunctionMode.Destroy();
		m_customExpressionsOnFunctionMode = null;
		m_staticSwitchNodes.Destroy();
		m_staticSwitchNodes = null;
		m_functionInputNodes.Destroy();
		m_functionInputNodes = null;
		m_functionNodes.Destroy();
		m_functionNodes = null;
		m_functionOutputNodes.Destroy();
		m_functionOutputNodes = null;
		m_functionSwitchNodes.Destroy();
		m_functionSwitchNodes = null;
		m_functionSwitchCopyNodes.Destroy();
		m_functionSwitchCopyNodes = null;
		m_multiPassMasterNodes.Destroy();
		m_multiPassMasterNodes = null;
		for( int i = 0; i < m_lodMultiPassMasterNodes.Count; i++ )
		{
			m_lodMultiPassMasterNodes[ i ].Destroy();
			m_lodMultiPassMasterNodes[ i ] = null;
		}
		m_lodMultiPassMasterNodes.Clear();
		m_lodMultiPassMasterNodes = null;
		m_texturePropertyNodes.Destroy();
		m_texturePropertyNodes = null;
		m_textureArrayNodes.Destroy();
		m_textureArrayNodes = null;
		m_screenColorNodes.Destroy();
		m_screenColorNodes = null;
		m_localVarNodes.Destroy();
		m_localVarNodes = null;
		m_globalArrayNodes.Destroy();
		m_globalArrayNodes = null;
		m_selectedNodes.Clear();
		m_selectedNodes = null;
		m_markedForDeletion.Clear();
		m_markedForDeletion = null;
		m_nodesDict.Clear();
		m_nodesDict = null;
		m_nodePreviewList.Clear();
		m_nodePreviewList = null;
		IsDirty = true;
		OnNodeEvent = null;
		OnDuplicateEvent = null;
		OnMaterialUpdatedEvent = null;
		OnShaderUpdatedEvent = null;
		OnEmptyGraphDetectedEvt = null;
		nodeStyleOff = null;
		nodeStyleOn = null;
		nodeTitle = null;
		commentaryBackground = null;
		OnLODMasterNodesAddedEvent = null;
	}
	void OnNodeChangeSizeEvent( ParentNode node )
	{
		m_nodeGrid.RemoveNodeFromGrid( node, true );
		m_nodeGrid.AddNodeToGrid( node );
	}
	public void OnNodeFinishMoving( ParentNode node, bool testOnlySelected, InteractionMode interactionMode )
	{
		if( OnNodeEvent != null )
		{
			OnNodeEvent( node );
			SaveIsDirty = true;
		}
		m_nodeGrid.RemoveNodeFromGrid( node, true );
		m_nodeGrid.AddNodeToGrid( node );
		{
			for( int i = m_nodes.Count - 1; i > -1; i-- )
			{
				if( node.UniqueId != m_nodes[ i ].UniqueId )
				{
					switch( interactionMode )
					{
						case InteractionMode.Target:
						{
							node.OnNodeInteraction( m_nodes[ i ] );
						}
```

```
                                                break;
                                                case InteractionMode.Other:
                                                {
                                                        m_nodes[ i ].OnNodeInteraction( node );
                                                }
                                                break;
                                                case InteractionMode.Both:
                                                {
                                                        node.OnNodeInteraction( m_nodes[ i ] );
                                                        m_nodes[ i ].OnNodeInteraction( node );
                                                }
                                                break;
                                        }
                                }
                        }
                }
        }
public void OnNodeReOrderEvent( ParentNode node, int index )
{
        if( node.Depth < index )
        {
                Debug.LogWarning( "Reorder canceled: This is a specific method for when reordering needs to be done and a its original index is higher than the new one" );
        }
        else
        {
                m_nodes.Remove( node );
                m_nodes.Insert( index, node );
                m_markToReOrder = true;
        }
}
public void AddNode( ParentNode node, bool updateId = false, bool addLast = true, bool registerUndo = true, bool fetchMaterialValues = true )
{
        if( registerUndo )
        {
                UIUtils.MarkUndoAction();
                Undo.RegisterCompleteObjectUndo( ParentWindow, Constants.UndoCreateNodeId );
                Undo.RegisterCompleteObjectUndo( this, Constants.UndoCreateNodeId );
                Undo.RegisterCreatedObjectUndo( node, Constants.UndoCreateNodeId );
        }
        if( OnNodeEvent != null )
        {
                OnNodeEvent( node );
        }
        if( updateId )
        {
                node.UniqueId = GetValidId();
        }
        else
        {
                UpdateIdFromNode( node );
        }
        if( addLast )
        {
                m_nodes.Add( node );
                node.Depth = m_nodes.Count;
        }
        else
        {
                m_nodes.Insert( 0, node );
                node.Depth = 0;
        }
        if( m_nodesDict.ContainsKey( node.UniqueId ) )
        {
                m_foundDuplicates = true;
        }
        else
        {
                m_nodesDict.Add( node.UniqueId, node );
                node.SetMaterialMode( CurrentMaterial, fetchMaterialValues );
        }
        m_nodeGrid.AddNodeToGrid( node );
        node.OnNodeChangeSizeEvent += OnNodeChangeSizeEvent;
        node.OnNodeReOrderEvent += OnNodeReOrderEvent;
        IsDirty = true;
}
public void CheckForDuplicates()
{
        if( m_foundDuplicates )
        {
                Debug.LogWarning( "Found duplicates:" );
                m_foundDuplicates = false;
                m_nodesDict.Clear();
                int count = m_nodes.Count;
                for( int i = 0; i < count; i++ )
                {
                        if( m_nodesDict.ContainsKey( m_nodes[ i ].UniqueId ) )
                        {
                                m_nodes[ i ].UniqueId = GetValidId();
                                m_nodesDict.Add( m_nodes[ i ].UniqueId, m_nodes[ i ] );
                                Debug.LogWarning( "Assigning new ID to " + m_nodes[ i ].TypeName );
                        }
                        else
                        {
```

```
                                        m_nodesDict.Add( m_nodes[ i ].UniqueId, m_nodes[ i ] );
                                }
                        }
                }
        }
        public ParentNode GetClickedNode()
        {
                if( m_nodeClicked < 0 )
                        return null;
                return GetNode( m_nodeClicked );
        }
        public PropertyNode GetInternalTemplateNode( int nodeId )
        {
                if( m_internalTemplateNodesDict.Count != m_internalTemplateNodesList.Count )
                {
                        m_internalTemplateNodesDict.Clear();
                        int count = m_internalTemplateNodesList.Count;
                        for( int i = 0; i < m_internalTemplateNodesList.Count; i++ )
                        {
                                if( m_internalTemplateNodesList[ i ] != null )
                                        m_internalTemplateNodesDict.Add( m_internalTemplateNodesList[ i ].UniqueId, m_internalTemplateNodesList[ i ] );
                        }
                }
                if( m_internalTemplateNodesDict.ContainsKey( nodeId ) )
                        return m_internalTemplateNodesDict[ nodeId ];
                return null;
        }
        public PropertyNode GetInternalTemplateNode( string propertyName )
        {
                return m_internalTemplateNodesList.Find( ( x ) => x.PropertyName.Equals( propertyName ) );
        }
        public void AddInternalTemplateNode( TemplateShaderPropertyData data )
        {
                PropertyNode propertyNode = null;
                switch( data.PropertyDataType )
                {
                        case WirePortDataType.FLOAT:
                        propertyNode = CreateInstance<RangedFloatNode>(); break;
                        case WirePortDataType.FLOAT4:
                        propertyNode = CreateInstance<Vector4Node>();
                        break;
                        case WirePortDataType.COLOR:
                        propertyNode = CreateInstance<ColorNode>();
                        break;
                        case WirePortDataType.INT:
                        propertyNode = CreateInstance<IntNode>(); break;
                        case WirePortDataType.SAMPLER1D:
                        case WirePortDataType.SAMPLER2D:
                        case WirePortDataType.SAMPLER3D:
                        case WirePortDataType.SAMPLERCUBE:
                        case WirePortDataType.SAMPLER2DARRAY:
                        propertyNode = CreateInstance<SamplerNode>();
                        break;
                        default: return;
                }
                propertyNode.PropertyNameFromTemplate( data );
                int uniqueId = -( m_internalTemplateNodesList.Count + 2 );
                propertyNode.SetBaseUniqueId( uniqueId );
                if( data.PropertyDataType == WirePortDataType.FLOAT ||
                        data.PropertyDataType == WirePortDataType.INT )
                        m_floatNodes.AddNode( propertyNode );
                m_internalTemplateNodesList.Add( propertyNode );
                m_internalTemplateNodesDict.Add( uniqueId, propertyNode );
        }
        public void ClearInternalTemplateNodes()
        {
                if( m_internalTemplateNodesList != null )
                {
                        int count = m_internalTemplateNodesList.Count;
                        for( int i = 0; i < count; i++ )
                        {
                                m_internalTemplateNodesList[ i ].Destroy();
                                GameObject.DestroyImmediate( m_internalTemplateNodesList[ i ] );
                        }
                        m_internalTemplateNodesList.Clear();
                        m_internalTemplateNodesDict.Clear();
                }
        }
        public ParentNode GetNode( int nodeId )
        {
                if( m_nodesDict.Count != m_nodes.Count )
                {
                        m_nodesDict.Clear();
                        int count = m_nodes.Count;
                        for( int i = 0; i < count; i++ )
                        {
                                if( m_nodes[ i ] != null && !m_nodesDict.ContainsKey( m_nodes[ i ].UniqueId ) )
                                        m_nodesDict.Add( m_nodes[ i ].UniqueId, m_nodes[ i ] );
                        }
                }
                if( m_nodesDict.ContainsKey( nodeId ) )
                        return m_nodesDict[ nodeId ];
                return null;
```

```
	}
	public void ForceReOrder()
	{
		m_nodes.Sort( ( x, y ) => x.Depth.CompareTo( y.Depth ) );
	}
	public bool Draw( DrawInfo drawInfo )
	{
		MasterNode masterNode = GetNode( m_masterNodeId ) as MasterNode;
		if( m_forceCategoryRefresh && masterNode != null )
		{
			masterNode.RefreshAvailableCategories();
			m_forceCategoryRefresh = false;
		}
		SaveIsDirty = false;
		if( m_afterDeserializeFlag )
		{
			CleanCorruptedNodes();
			if( m_nodes.Count == 0 )
			{
				NodeAvailability cachedCanvas = CurrentCanvasMode;
				ParentWindow.CreateNewGraph( "Empty" );
				CurrentCanvasMode = cachedCanvas;
				if( OnEmptyGraphDetectedEvt != null )
				{
					OnEmptyGraphDetectedEvt( this );
					SaveIsDirty = false;
				}
				else
				{
					SaveIsDirty = true;
				}
			}
		}
		if( drawInfo.CurrentEventType == EventType.Repaint )
		{
			if( m_markedToDeSelect )
				DeSelectAll();
			if( m_markToSelect > -1 )
			{
				AddToSelectedNodes( GetNode( m_markToSelect ) );
				m_markToSelect = -1;
			}
			if( m_markToReOrder )
			{
				m_markToReOrder = false;
				int nodesCount = m_nodes.Count;
				for( int i = 0; i < nodesCount; i++ )
				{
					m_nodes[ i ].Depth = i;
				}
			}
		}
		if( drawInfo.CurrentEventType == EventType.Repaint )
		{
			NodeLOD newLevel = NodeLOD.LOD0;
			float referenceValue;
			if( drawInfo.InvertedZoom > 0.5f )
			{
				newLevel = NodeLOD.LOD0;
				referenceValue = 4;
			}
			else if( drawInfo.InvertedZoom > 0.25f )
			{
				newLevel = NodeLOD.LOD1;
				referenceValue = 2;
			}
			else if( drawInfo.InvertedZoom > 0.15f )
			{
				newLevel = NodeLOD.LOD2;
				referenceValue = 1;
			}
			else if( drawInfo.InvertedZoom > 0.1f )
			{
				newLevel = NodeLOD.LOD3;
				referenceValue = 0;
			}
			else if( drawInfo.InvertedZoom > 0.07f )
			{
				newLevel = NodeLOD.LOD4;
				referenceValue = 0;
			}
			else
			{
				newLevel = NodeLOD.LOD5;
				referenceValue = 0;
			}
			nodeStyleOff = UIUtils.GetCustomStyle( CustomStyle.NodeWindowOff );
			nodeStyleOn = UIUtils.GetCustomStyle( CustomStyle.NodeWindowOn );//= UIUtils.GetCustomStyle( CustomStyle.NodeWindowOn );
			nodeTitle = UIUtils.GetCustomStyle( CustomStyle.NodeHeader );
			commentaryBackground = UIUtils.GetCustomStyle( CustomStyle.CommentaryBackground );
			if( newLevel != m_lodLevel || ( UIUtils.MainSkin != null && UIUtils.MainSkin.textField.border.left != referenceValue ) )
			{
				m_lodLevel = newLevel;
```