靡不有初，鲜克有终
» CSharpGL @ GitHub
» 我的博客汇总
» 捐赠列表
加入QQ群

昵称：BIT祝威
园龄：5年
荣誉：推荐博客
粉丝：624
关注：12
+加关注

最新随笔

1. CSharpGL(40)一种极其简单的半透明渲染方法

2. CSharpGL(39)GLSL光照示例：鼠标拖动太阳（光源）观察平行光的漫反射和镜面反射效果

3. CSharpGL(38)带初始数据创建Vertex Buffer Object的情形汇总

4. CSharpGL(37)创建和使用VBO的最佳方式

5. CSharpGL(36)通用的非托管数组排序方法

6. CSharpGL(35)用ViewPort实现类似3DMax那样的把一个场景渲染到4个视口

7. CSharpGL(34)以从零编写一个KleinBottle渲染器为例学习如何使用CSharpGL

8. CSharpGL(33)使用uniform块来优化对uniform变量的读写

9. CSharpGL(32)矩阵与四元数与角度旋转轴的相互转换

10. CSharpGL(31)[译]OpenGL渲染管道那些事

最新评论

1. Re:C#自定义控件：WinForm将其它应用程序窗体嵌入自己内部

# Unity3D核心类型一览

## 阅读目录(Content)

## Unity3D核心类型一览

@vczz Action appIdleAction = null; EventHandler appIdleEvent = null; public App......

--39

2. Re:C#自定义控件：WinForm将其它应用程序窗体嵌入自己内部

@
好的，我自己再研究下下。

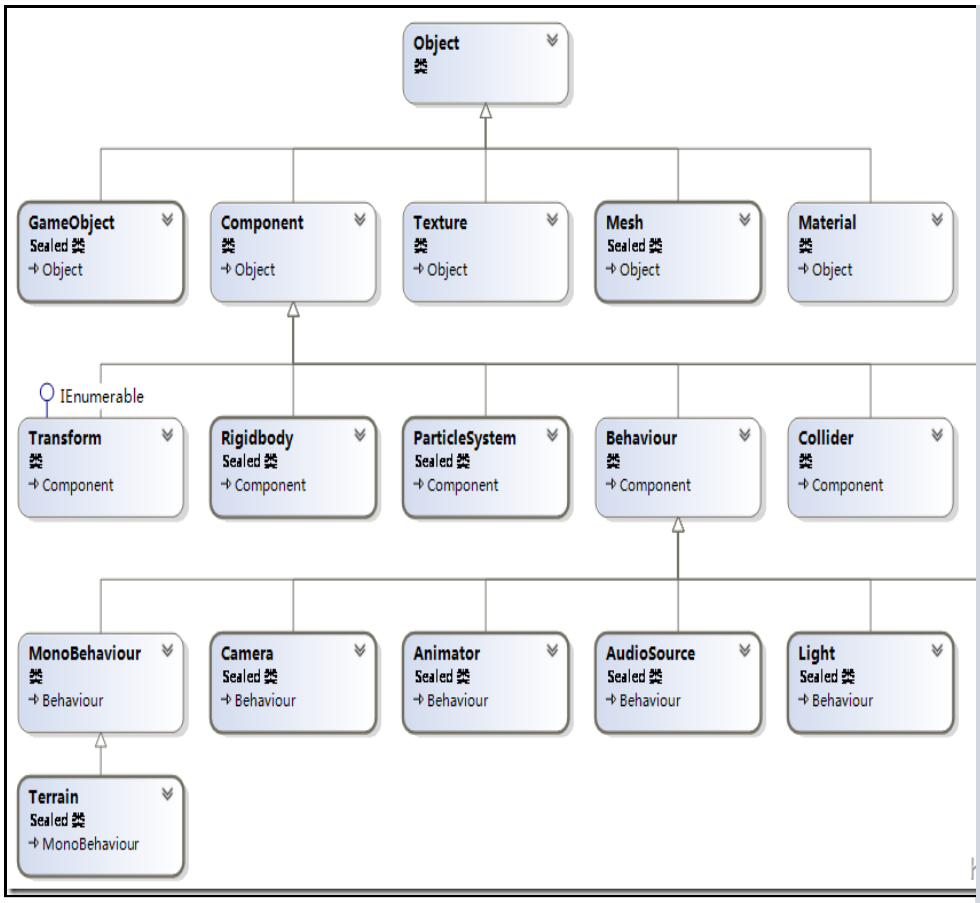
--vczz

3. Re:C#自定义控件：WinForm将其它应用程序窗体嵌入自己内部

@vczz引用 祝威你好，又是我，不好意思又来打扰你了。我在用你的软件时，发现如果嵌入的窗口比较大的话，往往显示的位置很奇怪，除非最大化，不然可能都看不到要嵌入的窗口在哪里（最大化后才能看到一部......

--BIT祝威

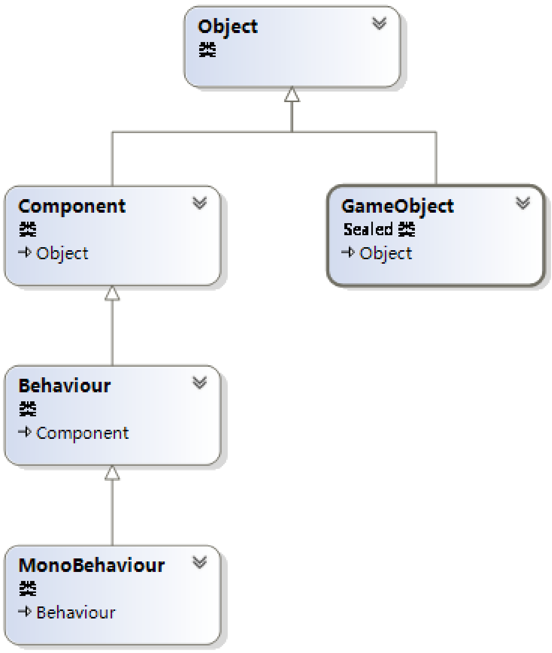
4. Re:C#自定义控件：WinForm将其它应用程序窗体嵌入自己内部



本文记录了Unity3D的最基本的核心类型。包括Object、GameObject、Component、Transform、Behaviour、Renderer、Collider、Rigidbody、Camera、Light、MonoBehaviour等。

需要展开了public类型方法的类图请点这里（http://www.cnblogs.com/bitzhuwei/gallery/image/152116.html）。

最核心的类型就这几个：Object、GameObject、Component、Behaviour、MonoBehaviour。

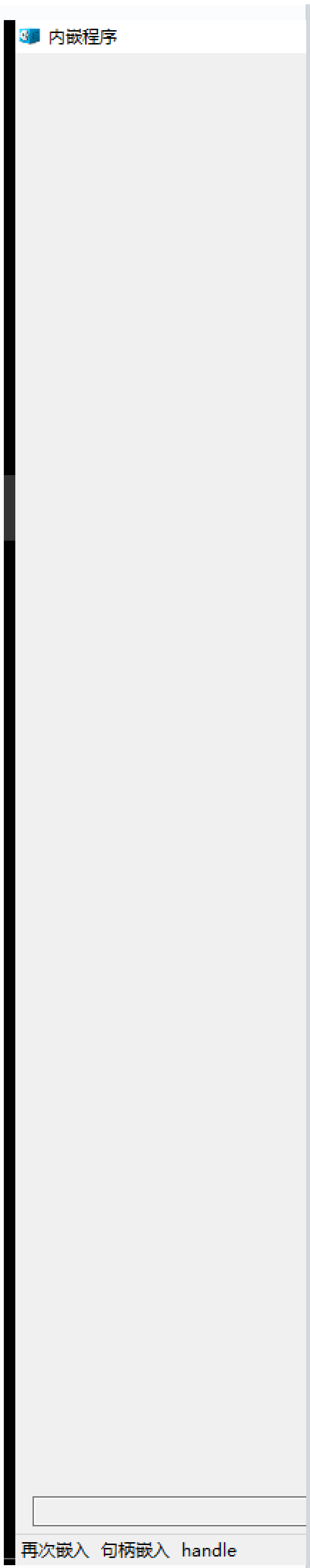


需要展开了这几个public类型方法的类图请点这里（http://www.cnblogs.com/bitzhuwei/gallery/ima

推荐：1

邵(g o)

## UnityEngine.Object



--vczz

5. Re:C#自定义控件：WinForm将其它应用程序窗体嵌入自己内部

祝威你好，又是我，不好意思又来打扰你了。我在用你的软件时，发现如果嵌入的窗口比较大的话，往往显示的位置很奇怪，除非最大化，不然可能都看不到要嵌入的窗口在哪里（最大化后才能看到一部分），而我在看你的源码......

所有Unity3D的基类。

持有实例的ID信息。

实现了静态方法：增（Instantiate）删（Destroy）查（FindObjectsOfType）

Any public variable you make that derives from Object gets shown in the inspector as a drop target, allowing you to set the value from the GUI.

View Code

回到顶部(go to top)

## UnityEngine.GameObject

/// <summary>
/// game object contains components.
/// <para>Add Component</para>
/// <para>Find Component</para>
/// <para>common components</para>
/// <para>BroadcastMessage在这个游戏物体及其子物体的所有MonoBehaviour中调用名称为methodName的方法.</para>
/// </summary>

`GameObject.active` is obsolete. Use `GameObject.SetActive()` , `GameObject.activeSelf` (read only) or `GameObject.activeInHierarchy` (read only) .

`gameObject.SetActiveRecursively()` is obsolete. Use `GameObject.SetActive()` , which is now inherited by children.

View Code

回到顶部(go to top)

## UnityEngine.Component

所有的Component，都会指向其所属的GameObject。

在脚本中用 `this.renderer` ， `this.transform` ， `this.GetComponent(XXX)` ， `this.gameObject` 与 `this.gameObject.renderer` ， `this.gameObject.transform` ， `this.gameObject.GetComponent(XXX)` ， `this.gameObject.gameObject` 的结果是完全一样的。这意味着，你用 `this.renderer.transform.renderer.collider` 这种写法，仍然可以得到 `this.collider` 。（在这些组件不是 `null` 的前提下）

the `active` property is deprecated on components. Please use `gameObject.active` instead. If you meant to enable / disable a single component use `enabled` instead.

`GameObject.active` is obsolete. Use `GameObject.SetActive()`, `GameObject.activeSelf` (read only) or `GameObject.activeInHierarchy` (read only) .

View Code

```
        //this.gameObject.active = false;//GameObject.active is obselete
        this.gameObject.SetActive(false);// ! use
        this.gameObject.activeSelf = false;//reado
        this.gameObject.activeInHierarchy = false;

```

推荐: 1

--vczz

## 阅读排行榜

1. C#自定义控件：WinForm将其它应用程序窗体嵌入自己内部(21342)
2. 继电器是如何成为CPU的（1）(19476)
3. 图文详解Unity3D中Material的Tiling和Offset是怎么回事(9054)
4. 《30天自制操作系统》笔记(01)——hello bitzhuwei's OS!(8775)
5. Unity3D核心类型一览(8597)
6. 《30天自制操作系统》笔记(02)——导入C语言(8075)
7. SharpGL(Opengl)入门之纹理星球(7181)
8. 一个编译器的实现0(7176)
9. [译+改]最长回文子串(Longest Palindromic Substring) Part II(7049)
10. 《30天自制操作系统》笔记(03)——使用Vmware(6077)
11. [Unity3D入门]分享一个自制的入门级游戏项目"坦克狙击手"(6034)
12. 《30天自制操作系统》笔记(12)——多任务入门(5661)
13. CSharpGL(0)一个易学易用的C#版OpenGL(5576)
14. OpenGL入门暨用C#做个3D吞食鱼（一）第一人称视角的实现(5504)
15. 【OpenGL(SharpGL)】支持任意相机可平移缩放的轨迹球实现(4812)

## 评论排行榜

1. C#自定义控件：WinForm将其它应用程序窗体嵌入自己内部(190)
2. 继电器是如何成为CPU的（1）(167)
3. [Unity3D入门]入门级游戏项目"坦克狙击手"更新(103)
4. [Unity3D入门]分享一个自制的入门级游戏项目"坦克狙击手"(103)
5. CSharpGL(0)一个易学易用的C#版OpenGL(48)

```
        //this.active = false;//Component.active is obsolete
        this.transform.active = false;//cannot disable singly
        this.particleSystem.active = false;//cannot disable singly
        this.rigidbody.active = false;//cannot disable singly

        this.GetComponent<TestEqual>().enabled = false;//work on single behaviour
        this.renderer.enabled = false;//work on single renderer
        this.collider.enabled = false;//work on single collider
```

回到顶部(go to top)

## UnityEngine.Texture

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;

    public class Texture : UnityEngine.Object
    {
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern int GetNativeTextureID();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern IntPtr GetNativeTexturePtr();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern int Internal_GetHeight(Texture mono);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void Internal_GetTexelSize(Texture tex, out Vector2 output);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern int Internal_GetWidth(Texture mono);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public static extern void SetGlobalAnisotropicFilteringLimits(int forcedMin, int globalMax);

        public int anisoLevel { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public static AnisotropicFiltering anisotropicFiltering { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public FilterMode filterMode { [MethodImpl(MethodImplOptions.Internal
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessI
set; }

        public virtual int height
        {
            get
            {
```

推荐: 1

6. [我给Unity官方视频教程做中文字幕]beginner Graphics – Lessons系列之摄像机介绍Cameras(39)

7. 继电器是如何成为CPU的(2)(33)

8. [我给Unity官方视频教程做中文字幕]beginner Graphics – Lessons系列之网格渲染器和过滤器Mesh renderers and filters(31)

9. 《30天自制操作系统》笔记(02)——导入C语言(29)

10. 《30天自制操作系统》笔记(01)——hello bitzhuwei's OS!(27)

11. [我给Unity官方视频教程做中文字幕]beginner Graphics – Lessons系列之材质了解Materials(24)

12. 用C表达面向对象语言的机制——C#版(24)

13. [我给Unity官方视频教程做中文字幕]beginner Graphics – Lessons系列之纹理Textures(23)

14. [我给Unity官方视频教程做中文字幕]beginner Graphics – Lessons系列之灯光介绍Lights(18)

15. SharpGL(Opengl)入门之纹理星球(18)

```
                return Internal_GetHeight(this);
            }
            set
            {
                throw new Exception("not implemented");
            }
        }

        public static int masterTextureLimit {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float mipMapBias { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

        public Vector2 texelSize
        {
            get
            {
                Vector2 vector;
                Internal_GetTexelSize(this, out vector);
                return vector;
            }
        }

        public virtual int width
        {
            get
            {
                return Internal_GetWidth(this);
            }
            set
            {
                throw new Exception("not implemented");
            }
        }

        public TextureWrapMode wrapMode {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(go to top)

## UnityEngine.Mesh

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
```

推荐: 1

```
using System.Runtime.InteropServices;
using UnityEngine.Internal;

public sealed class Mesh : UnityEngine.Object
{
    public Mesh()
    {
        Internal_Create(this);
    }

    [ExcludeFromDocs]
    public void Clear()
    {
        bool keepVertexLayout = true;
        this.Clear(keepVertexLayout);
    }

    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern void Clear([DefaultValue("true")] bool keepVertexLayout);
    [ExcludeFromDocs]
    public void CombineMeshes(CombineInstance[] combine)
    {
        bool useMatrices = true;
        bool mergeSubMeshes = true;
        this.CombineMeshes(combine, mergeSubMeshes, useMatrices);
    }

    [ExcludeFromDocs]
    public void CombineMeshes(CombineInstance[] combine, bool mergeSubMeshes)
    {
        bool useMatrices = true;
        this.CombineMeshes(combine, mergeSubMeshes, useMatrices);
    }

    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern void CombineMeshes(CombineInstance[] combine, [DefaultValue("true")] bool mergeSubMeshes, [DefaultValue("true")] bool useMatrices);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern int GetBlendShapeIndex(string blendShapeName);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern string GetBlendShapeName(int index);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern int[] GetIndices(int submesh);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern MeshTopology GetTopology(int submesh);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern int[] GetTriangles(int submesh);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall, Obsolete("Use GetTriangles instead. Internally this function converts a list of triangles to a strip, so it might be slow, it might be a mess.")]
    public extern int[] GetTriangleStrip(int submesh);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void Internal_Create([Writable] Mesh mono);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_bounds(
    [MethodImpl(MethodImplOptions.InternalCal
    private extern void INTERNAL_set_bounds(
    [MethodImpl(MethodImplOptions.InternalCal
```

推荐: 1

```
        public extern void MarkDynamic();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Optimize();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void RecalculateBounds();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void RecalculateNormals();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetIndices(int[] indices, MeshTopology topology, int submesh);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetTriangles(int[] triangles, int submesh);
        [MethodImpl(MethodImplOptions.InternalCall), Obsolete("Use SetTriangles instead. Internally this function will convert the triangle strip to a list of triangles anyway."), WrapperlessIcall]
        public extern void SetTriangleStrip(int[] triangles, int submesh);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void UploadMeshData(bool markNoLogerReadable);

        public Matrix4x4[] bindposes { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int blendShapeCount { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public BoneWeight[] boneWeights { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Bounds bounds
        {
            get
            {
                Bounds bounds;
                this.INTERNAL_get_bounds(out bounds);
                return bounds;
            }
            set
            {
                this.INTERNAL_set_bounds(ref value);
            }
        }

        internal bool canAccess { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Color[] colors { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Color32[] colors32 { [MethodImpl(MethodImplOptions.InternalCa
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessI
set; }

        public bool isReadable { [MethodImpl(Meth                    all)
WrapperlessIcall] get; }

```

推荐: 1

```
        public Vector3[] normals { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int subMeshCount { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector4[] tangents { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int[] triangles { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector2[] uv { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector2[] uv1
        {
            get
            {
                return this.uv2;
            }
            set
            {
                this.uv2 = value;
            }
        }

        public Vector2[] uv2 { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int vertexCount { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Vector3[] vertices { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(go to top)

## UnityEngine.Material

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;
    using UnityEngine.Internal;

    public class Material : UnityEngine.Object
    {
        public Material(string contents)
        {
            Internal_CreateWithString(this, contents);
        }

        public Material(Material source)
        {
            Internal_CreateWithMaterial(this, source);
        }

        public Material(Shader shader)
        {
            Internal_CreateWithShader(this, shader);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void CopyPropertiesFromMaterial(Material mat);
        [Obsolete("Use the Material constructor instead.")]
        public static Material Create(string scriptContents)
        {
            return new Material(scriptContents);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void DisableKeyword(string keyword);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void EnableKeyword(string keyword);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern Color GetColor(int nameID);
        public Color GetColor(string propertyName)
        {
            return this.GetColor(Shader.PropertyToID(propertyName));
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern float GetFloat(int nameID);
        public float GetFloat(string propertyName)
        {
            return this.GetFloat(Shader.PropertyToID(propertyName));
        }

        public int GetInt(int nameID)
        {
            return (int) this.GetFloat(nameID);
        }

        public int GetInt(string propertyName)
        {
            return (int) this.GetFloat(propertyNa
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
```

推荐: 1

```
        public extern Matrix4x4 GetMatrix(int nameID);
        public Matrix4x4 GetMatrix(string propertyName)
        {
            return this.GetMatrix(Shader.PropertyToID(propertyName));
        }

        [ExcludeFromDocs]
        public string GetTag(string tag, bool searchFallbacks)
        {
            string defaultValue = string.Empty;
            return this.GetTag(tag, searchFallbacks, defaultValue);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern string GetTag(string tag, bool searchFallbacks,
[DefaultValue("\"\"")] string defaultValue);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern Texture GetTexture(int nameID);
        public Texture GetTexture(string propertyName)
        {
            return this.GetTexture(Shader.PropertyToID(propertyName));
        }

        public Vector2 GetTextureOffset(string propertyName)
        {
            Vector2 vector;
            Internal_GetTextureOffset(this, propertyName, out vector);
            return vector;
        }

        public Vector2 GetTextureScale(string propertyName)
        {
            Vector2 vector;
            Internal_GetTextureScale(this, propertyName, out vector);
            return vector;
        }

        public Vector4 GetVector(int nameID)
        {
            Color color = this.GetColor(nameID);
            return new Vector4(color.r, color.g, color.b, color.a);
        }

        public Vector4 GetVector(string propertyName)
        {
            Color color = this.GetColor(propertyName);
            return new Vector4(color.r, color.g, color.b, color.a);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern bool HasProperty(int nameID);
        public bool HasProperty(string propertyName)
        {
            return this.HasProperty(Shader.PropertyToID(propertyName));
        }

        [MethodImpl(MethodImplOptions.InternalCal
        private static extern void INTERNAL_CALL_SetColor(Material self, int nameID,
ref Color color);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_SetMatrix(Material self, int nameID, ref Matrix4x4 matrix);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_SetTextureOffset(Material self, string propertyName, ref Vector2 offset);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_SetTextureScale(Material self, string propertyName, ref Vector2 scale);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void Internal_CreateWithMaterial([Writable] Material mono, Material source);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void Internal_CreateWithShader([Writable] Material mono, Shader shader);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void Internal_CreateWithString([Writable] Material mono, string contents);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void Internal_GetTextureOffset(Material mat, string name, out Vector2 output);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void Internal_GetTextureScale(Material mat, string name, out Vector2 output);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Lerp(Material start, Material end, float t);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetBuffer(string propertyName, ComputeBuffer buffer);
        public void SetColor(int nameID, Color color)
        {
            INTERNAL_CALL_SetColor(this, nameID, ref color);
        }

        public void SetColor(string propertyName, Color color)
        {
            this.SetColor(Shader.PropertyToID(propertyName), color);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetFloat(int nameID, float value);
        public void SetFloat(string propertyName, float value)
        {
            this.SetFloat(Shader.PropertyToID(propertyName), value);
        }

        public void SetInt(int nameID, int value)
        {
            this.SetFloat(nameID, (float) value);
        }

        public void SetInt(string propertyName, int value)
        {
            this.SetFloat(propertyName, (float) value);
        }

        public void SetMatrix(int nameID, Matrix4
        {
```

推荐: 1

```
            INTERNAL_CALL_SetMatrix(this, nameID, ref matrix);
        }

        public void SetMatrix(string propertyName, Matrix4x4 matrix)
        {
            this.SetMatrix(Shader.PropertyToID(propertyName), matrix);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern bool SetPass(int pass);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetTexture(int nameID, Texture texture);
        public void SetTexture(string propertyName, Texture texture)
        {
            this.SetTexture(Shader.PropertyToID(propertyName), texture);
        }

        public void SetTextureOffset(string propertyName, Vector2 offset)
        {
            INTERNAL_CALL_SetTextureOffset(this, propertyName, ref offset);
        }

        public void SetTextureScale(string propertyName, Vector2 scale)
        {
            INTERNAL_CALL_SetTextureScale(this, propertyName, ref scale);
        }

        public void SetVector(int nameID, Vector4 vector)
        {
            this.SetColor(nameID, new Color(vector.x, vector.y, vector.z, vector.w));
        }

        public void SetVector(string propertyName, Vector4 vector)
        {
            this.SetColor(propertyName, new Color(vector.x, vector.y, vector.z, vector.w));
        }

        public Color color
        {
            get
            {
                return this.GetColor("_Color");
            }
            set
            {
                this.SetColor("_Color", value);
            }
        }

        public Texture mainTexture
        {
            get
            {
                return this.GetTexture("_MainTex"
            }
            set
            {
                this.SetTexture("_MainTex", value);
            }
        }

        public Vector2 mainTextureOffset
        {
            get
            {
                return this.GetTextureOffset("_MainTex");
            }
            set
            {
                this.SetTextureOffset("_MainTex", value);
            }
        }

        public Vector2 mainTextureScale
        {
            get
            {
                return this.GetTextureScale("_MainTex");
            }
            set
            {
                this.SetTextureScale("_MainTex", value);
            }
        }

        public int passCount { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public int renderQueue { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Shader shader { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public string[] shaderKeywords { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(go to top)

## UnityEngine.Transform

```
namespace UnityEngine
{
```

```
using System;
using System.Collections;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using UnityEngine.Internal;

public class Transform : Component, IEnumerable
{
    protected Transform()
    {
    }

    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern void DetachChildren();
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern Transform Find(string name);
    public Transform FindChild(string name)
    {
        return this.Find(name);
    }

    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern Transform GetChild(int index);
    [MethodImpl(MethodImplOptions.InternalCall), Obsolete("use Transform.childCount instead."), WrapperlessIcall]
    public extern int GetChildCount();
    public IEnumerator GetEnumerator()
    {
        return new Enumerator(this);
    }

    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern int GetSiblingIndex();
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Vector3 INTERNAL_CALL_InverseTransformDirection(Transform self, ref Vector3 direction);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Vector3 INTERNAL_CALL_InverseTransformPoint(Transform self, ref Vector3 position);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Vector3 INTERNAL_CALL_InverseTransformVector(Transform self, ref Vector3 vector);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void INTERNAL_CALL_LookAt(Transform self, ref Vector3 worldPosition, ref Vector3 worldUp);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void INTERNAL_CALL_RotateAround(Transform self, ref Vector3 axis, float angle);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void INTERNAL_CALL_RotateAroundInternal(Transform self, ref Vector3 axis, float angle);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void INTERNAL_CALL_RotateAroundLocal(Transform
ref Vector3 axis, float angle);
    [MethodImpl(MethodImplOptions.InternalCal
    private static extern Vector3 INTERNAL_CA                          ans
self, ref Vector3 direction);
    [MethodImpl(MethodImplOptions.InternalCal
    private static extern Vector3 INTERNAL_CALL_TransformPoint(Transform self, ref Vector3 position);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Vector3 INTERNAL_CALL_TransformVector(Transform self, ref Vector3 vector);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_localEulerAngles(out Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_localPosition(out Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_localRotation(out Quaternion value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_localScale(out Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_localToWorldMatrix(out Matrix4x4 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_lossyScale(out Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_position(out Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_rotation(out Quaternion value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_get_worldToLocalMatrix(out Matrix4x4 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_set_localEulerAngles(ref Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_set_localPosition(ref Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_set_localRotation(ref Quaternion value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_set_localScale(ref Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_set_position(ref Vector3 value);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private extern void INTERNAL_set_rotation(ref Quaternion value);
    public Vector3 InverseTransformDirection(Vector3 direction)
    {
        return INTERNAL_CALL_InverseTransformDirection(this, ref direction);
    }

    public Vector3 InverseTransformDirection(float x, float y, float z)
    {
        return this.InverseTransformDirection(new Vector3(x, y, z));
    }

    public Vector3 InverseTransformPoint(Vector3 position)
    {
        return INTERNAL_CALL_InverseTransformPoint(this, ref position);
    }

    public Vector3 InverseTransformPoint(float x, float y, float z)
    {
        return this.InverseTransformPoint(new Vector3(x, y, z));
    }

    public Vector3 InverseTransformVector(Vec
    {
        return INTERNAL_CALL_InverseTransform                  );
```

推荐: 1

```
        }

        public Vector3 InverseTransformVector(float x, float y, float z)
        {
            return this.InverseTransformVector(new Vector3(x, y, z));
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern bool IsChildOf(Transform parent);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern bool IsNonUniformScaleTransform();
        [ExcludeFromDocs]
        public void LookAt(Transform target)
        {
            Vector3 up = Vector3.up;
            this.LookAt(target, up);
        }

        [ExcludeFromDocs]
        public void LookAt(Vector3 worldPosition)
        {
            Vector3 up = Vector3.up;
            INTERNAL_CALL_LookAt(this, ref worldPosition, ref up);
        }

        public void LookAt(Transform target, [DefaultValue("Vector3.up")] Vector3 worldUp)
        {
            if (target != null)
            {
                this.LookAt(target.position, worldUp);
            }
        }

        public void LookAt(Vector3 worldPosition, [DefaultValue("Vector3.up")] Vector3 worldUp)
        {
            INTERNAL_CALL_LookAt(this, ref worldPosition, ref worldUp);
        }

        [ExcludeFromDocs]
        public void Rotate(Vector3 eulerAngles)
        {
            Space self = Space.Self;
            this.Rotate(eulerAngles, self);
        }

        [ExcludeFromDocs]
        public void Rotate(Vector3 axis, float angle)
        {
            Space self = Space.Self;
            this.Rotate(axis, angle, self);
        }

        public void Rotate(Vector3 eulerAngles, [[illegible]f")] relativeTo)
        {
            Quaternion quaternion = Quaternion.Euler(eulerAngles.x, eulerAngles.y,
```

推荐: 1

```
eulerAngles.z);
            if (relativeTo == Space.Self)
            {
                this.localRotation *= quaternion;
            }
            else
            {
                this.rotation *= (Quaternion.Inverse(this.rotation) * quaternion) * this.rotation;
            }
        }

        [ExcludeFromDocs]
        public void Rotate(float xAngle, float yAngle, float zAngle)
        {
            Space self = Space.Self;
            this.Rotate(xAngle, yAngle, zAngle, self);
        }

        public void Rotate(Vector3 axis, float angle, [DefaultValue("Space.Self")] Space relativeTo)
        {
            if (relativeTo == Space.Self)
            {
                this.RotateAroundInternal(base.transform.TransformDirection(axis), angle * 0.01745329f);
            }
            else
            {
                this.RotateAroundInternal(axis, angle * 0.01745329f);
            }
        }

        public void Rotate(float xAngle, float yAngle, float zAngle, [DefaultValue("Space.Self")] Space relativeTo)
        {
            this.Rotate(new Vector3(xAngle, yAngle, zAngle), relativeTo);
        }

        [Obsolete("use Transform.Rotate instead.")]
        public void RotateAround(Vector3 axis, float angle)
        {
            INTERNAL_CALL_RotateAround(this, ref axis, angle);
        }

        public void RotateAround(Vector3 point, Vector3 axis, float angle)
        {
            Vector3 position = this.position;
            Quaternion quaternion = Quaternion.AngleAxis(angle, axis);
            Vector3 vector2 = position - point;
            vector2 = (Vector3) (quaternion * vector2);
            position = point + vector2;
            this.position = position;
            this.RotateAroundInternal(axis, angle * 0.01745329f);
        }

        internal void RotateAroundInternal(Vector
        {
            INTERNAL_CALL_RotateAroundInternal(this, ref axis, angle);
        }

        [Obsolete("use Transform.Rotate instead.")]
        public void RotateAroundLocal(Vector3 axis, float angle)
        {
            INTERNAL_CALL_RotateAroundLocal(this, ref axis, angle);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern void SendTransformChangedScale();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetAsFirstSibling();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetAsLastSibling();
        public void SetParent(Transform parent)
        {
            this.SetParent(parent, true);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetParent(Transform parent, bool worldPositionStays);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetSiblingIndex(int index);
        public Vector3 TransformDirection(Vector3 direction)
        {
            return INTERNAL_CALL_TransformDirection(this, ref direction);
        }

        public Vector3 TransformDirection(float x, float y, float z)
        {
            return this.TransformDirection(new Vector3(x, y, z));
        }

        public Vector3 TransformPoint(Vector3 position)
        {
            return INTERNAL_CALL_TransformPoint(this, ref position);
        }

        public Vector3 TransformPoint(float x, float y, float z)
        {
            return this.TransformPoint(new Vector3(x, y, z));
        }

        public Vector3 TransformVector(Vector3 vector)
        {
            return INTERNAL_CALL_TransformVector(this, ref vector);
        }

        public Vector3 TransformVector(float x, float y, float z)
        {
            return this.TransformVector(new Vector3(x, y, z));
        }

        [ExcludeFromDocs]
        public void Translate(Vector3 translatio
        {
            Space self = Space.Self;
            this.Translate(translation, self);
        }

        public void Translate(Vector3 translation, [DefaultValue("Space.Self")] Space relativeTo)
        {
            if (relativeTo == Space.World)
            {
                this.position += translation;
            }
            else
            {
                this.position += this.TransformDirection(translation);
            }
        }

        public void Translate(Vector3 translation, Transform relativeTo)
        {
            if (relativeTo != null)
            {
                this.position += relativeTo.TransformDirection(translation);
            }
            else
            {
                this.position += translation;
            }
        }

        [ExcludeFromDocs]
        public void Translate(float x, float y, float z)
        {
            Space self = Space.Self;
            this.Translate(x, y, z, self);
        }

        public void Translate(float x, float y, float z, [DefaultValue("Space.Self")] Space relativeTo)
        {
            this.Translate(new Vector3(x, y, z), relativeTo);
        }

        public void Translate(float x, float y, float z, Transform relativeTo)
        {
            this.Translate(new Vector3(x, y, z), relativeTo);
        }

        public int childCount { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Vector3 eulerAngles
        {
            get
            {
                return this.rotation.eulerAngles;
            }
            set
            {
                this.rotation = Quaternion.Euler(value);
            }
        }

        public Vector3 forward
        {
            get
            {
                return (Vector3) (this.rotation * Vector3.forward);
            }
            set
            {
                this.rotation = Quaternion.LookRotation(value);
            }
        }

        public bool hasChanged { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector3 localEulerAngles
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_localEulerAngles(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_localEulerAngles(ref value);
            }
        }

        public Vector3 localPosition
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_localPosition(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_localPosition(ref value);
            }
        }

        public Quaternion localRotation
        {
            get
            {
                Quaternion quaternion;
                this.INTERNAL_get_localRotation(out quaternion);
                return quaternion;
            }
            set
            {
                this.INTERNAL_set_localRotation(ref value);
            }
        }

        public Vector3 localScale
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_localScale(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_localScale(ref value);
            }
        }

        public Matrix4x4 localToWorldMatrix
        {
            get
            {
                Matrix4x4 matrixx;
                this.INTERNAL_get_localToWorldMatrix(out matrixx);
                return matrixx;
            }
        }

        public Vector3 lossyScale
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_lossyScale(out vector);
                return vector;
            }
        }

        public Transform parent
        {
            get
            {
                return this.parentInternal;
            }
            set
            {
                if (this is RectTransform)
                {
                    Debug.LogWarning("Parent of RectTransform is being set with parent property. Consider using the SetParent method instead, with the worldPositionStays argument set to false. This will retain local orientation and scale rather than world orientation and scale, which can prevent common UI scaling issues.", this);
                }
                this.parentInternal = value;
            }
        }

        internal Transform parentInternal {
```

推荐: 1

```
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector3 position
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_position(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_position(ref value);
            }
        }

        public Vector3 right
        {
            get
            {
                return (Vector3) (this.rotation * Vector3.right);
            }
            set
            {
                this.rotation = Quaternion.FromToRotation(Vector3.right, value);
            }
        }

        public Transform root { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; }

        public Quaternion rotation
        {
            get
            {
                Quaternion quaternion;
                this.INTERNAL_get_rotation(out quaternion);
                return quaternion;
            }
            set
            {
                this.INTERNAL_set_rotation(ref value);
            }
        }

        public Vector3 up
        {
            get
            {
                return (Vector3) (this.rotation * Vector3.up);
            }
            set
            {
                this.rotation = Quaternion.FromTo                       ue)
            }
        }

```

推荐: 1

```
        public Matrix4x4 worldToLocalMatrix
        {
            get
            {
                Matrix4x4 matrixx;
                this.INTERNAL_get_worldToLocalMatrix(out matrixx);
                return matrixx;
            }
        }

        private sealed class Enumerator : IEnumerator
        {
            private int currentIndex = -1;
            private Transform outer;

            internal Enumerator(Transform outer)
            {
                this.outer = outer;
            }

            public bool MoveNext()
            {
                int childCount = this.outer.childCount;
                return (++this.currentIndex < childCount);
            }

            public void Reset()
            {
                this.currentIndex = -1;
            }

            public object Current
            {
                get
                {
                    return this.outer.GetChild(this.currentIndex);
                }
            }
        }
    }
}
```

回到顶部(go to top)

## UnityEngine.Renderer

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;

```

推荐: 1

```
    public class Renderer : Component
    {
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void GetPropertyBlock(MaterialPropertyBlock dest);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_lightmapTilingOffset(out Vector4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_localToWorldMatrix(out Matrix4x4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_worldToLocalMatrix(out Matrix4x4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_lightmapTilingOffset(ref Vector4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Render(int material);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetPropertyBlock(MaterialPropertyBlock properties);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern void SetSubsetIndex(int index, int subSetIndexForMaterial);

        public Bounds bounds { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool castShadows { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool enabled { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool isPartOfStaticBatch { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool isVisible { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public int lightmapIndex { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector4 lightmapTilingOffset
        {
            get
            {
                Vector4 vector;
                this.INTERNAL_get_lightmapTilingOffset(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_lightmapTilingOffset(ref value);
            }
        }

        public Transform lightProbeAnchor {
[MethodImpl(MethodImplOptions.InternalCall), Wrapperl
[MethodImpl(MethodImplOptions.InternalCall), Wrapperl

```

```
        public Matrix4x4 localToWorldMatrix
        {
            get
            {
                Matrix4x4 matrixx;
                this.INTERNAL_get_localToWorldMatrix(out matrixx);
                return matrixx;
            }
        }

        public Material material { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Material[] materials { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool receiveShadows { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Material sharedMaterial { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Material[] sharedMaterials { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int sortingLayerID { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public string sortingLayerName { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int sortingOrder { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        internal int staticBatchIndex { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        internal Transform staticBatchRootTransform { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool useLightProbes { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Matrix4x4 worldToLocalMatrix
        {
            get
            {
                Matrix4x4 matrixx;
                this.INTERNAL_get_worldToLocalMatrix(out matrixx);
                return matrixx;
            }
        }
    }
}
```

回到顶部(go to top)

## UnityEngine.ParticalSystem

```
namespace UnityEngine
{
    using System;
    using System.Collections;
    using System.Collections.Generic;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;
    using UnityEngine.Internal;

    public sealed class ParticleSystem : Component
    {
        [ExcludeFromDocs]
        public void Clear()
        {
            bool withChildren = true;
            this.Clear(withChildren);
        }

        public void Clear([DefaultValue("true")] bool withChildren)
        {
            if (withChildren)
            {
                foreach (ParticleSystem system in GetParticleSystems(this))
                {
                    system.Internal_Clear();
                }
            }
            else
            {
                this.Internal_Clear();
            }
        }

        public void Emit(int count)
        {
            INTERNAL_CALL_Emit(this, count);
        }

        public void Emit(Particle particle)
        {
            this.Internal_Emit(ref particle);
```

推荐: 1

```
        }

        public void Emit(Vector3 position, Vector3 velocity, float size, float lifetime, Color32 color)
        {
            Particle particle = new Particle {
                position = position,
                velocity = velocity,
                lifetime = lifetime,
                startLifetime = lifetime,
                size = size,
                rotation = 0f,
                angularVelocity = 0f,
                color = color,
                randomSeed = 5
            };
            this.Internal_Emit(ref particle);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern int GetCollisionEvents(GameObject go, CollisionEvent[] collisionEvents);
        private static void GetDirectParticleSystemChildrenRecursive(Transform transform, List<ParticleSystem> particleSystems)
        {
            IEnumerator enumerator = transform.GetEnumerator();
            try
            {
                while (enumerator.MoveNext())
                {
                    Transform current = (Transform) enumerator.Current;
                    ParticleSystem component = current.gameObject.GetComponent<ParticleSystem>();
                    if (component != null)
                    {
                        particleSystems.Add(component);
                        GetDirectParticleSystemChildrenRecursive(current, particleSystems);
                    }
                }
            }
            finally
            {
                IDisposable disposable = enumerator as IDisposable;
                if (disposable == null)
                {
                }
                disposable.Dispose();
            }
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern int GetParticles(Particle[] particles);
        internal static ParticleSystem[] GetParticleSystems(ParticleSystem r
        {
            if (root == null)
            {
                return null;
```

推荐: 1

```
            }
            List<ParticleSystem> particleSystems = new List<ParticleSystem> {
                root
            };
            GetDirectParticleSystemChildrenRecursive(root.transform, particleSystems);
            return particleSystems.ToArray();
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal static extern Collider InstanceIDToCollider(int instanceID);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_Emit(ParticleSystem self, int count);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_Clear();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_Emit(ref Particle particle);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_startColor(out Color value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool Internal_IsAlive();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_Pause();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_Play();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_startColor(ref Color value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_Simulate(float t, bool restart);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_Stop();
        [ExcludeFromDocs]
        public bool IsAlive()
        {
            bool withChildren = true;
            return this.IsAlive(withChildren);
        }

        public bool IsAlive([DefaultValue("true")] bool withChildren)
        {
            if (!withChildren)
            {
                return this.Internal_IsAlive();
            }
            foreach (ParticleSystem system in GetParticleSystems(this))
            {
                if (system.Internal_IsAlive())
                {
                    return true;
                }
            }
            return false;
        }

        [ExcludeFromDocs]
        public void Pause()
        {
```

推荐: 1

```
            bool withChildren = true;
            this.Pause(withChildren);
        }

        public void Pause([DefaultValue("true")] bool withChildren)
        {
            if (withChildren)
            {
                foreach (ParticleSystem system in GetParticleSystems(this))
                {
                    system.Internal_Pause();
                }
            }
            else
            {
                this.Internal_Pause();
            }
        }

        [ExcludeFromDocs]
        public void Play()
        {
            bool withChildren = true;
            this.Play(withChildren);
        }

        public void Play([DefaultValue("true")] bool withChildren)
        {
            if (withChildren)
            {
                foreach (ParticleSystem system in GetParticleSystems(this))
                {
                    system.Internal_Play();
                }
            }
            else
            {
                this.Internal_Play();
            }
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetParticles(Particle[] particles, int size);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern void SetupDefaultType(int type);
        [ExcludeFromDocs]
        public void Simulate(float t)
        {
            bool restart = true;
            bool withChildren = true;
            this.Simulate(t, withChildren, restart);
        }

        [ExcludeFromDocs]
        public void Simulate(float t, bool withCh
        {
            bool restart = true;
            this.Simulate(t, withChildren, restart);
        }

        public void Simulate(float t, [DefaultValue("true")] bool withChildren, [DefaultValue("true")] bool restart)
        {
            if (withChildren)
            {
                foreach (ParticleSystem system in GetParticleSystems(this))
                {
                    system.Internal_Simulate(t, restart);
                }
            }
            else
            {
                this.Internal_Simulate(t, restart);
            }
        }

        [ExcludeFromDocs]
        public void Stop()
        {
            bool withChildren = true;
            this.Stop(withChildren);
        }

        public void Stop([DefaultValue("true")] bool withChildren)
        {
            if (withChildren)
            {
                foreach (ParticleSystem system in GetParticleSystems(this))
                {
                    system.Internal_Stop();
                }
            }
            else
            {
                this.Internal_Stop();
            }
        }

        public float duration { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public float emissionRate { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool enableEmission { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float gravityModifier { [MethodImpl(MethodImplOptions.Interna
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessI
set; }

        public bool isPaused { [MethodImpl(Methoc                l),
WrapperlessIcall] get; }

```

推荐: 1

```
        public bool isPlaying { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool isStopped { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool loop { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int maxParticles { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int particleCount { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public float playbackSpeed { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool playOnAwake { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public uint randomSeed { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int safeCollisionEventSize { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public ParticleSystemSimulationSpace simulationSpace { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Color startColor
        {
            get
            {
                Color color;
                this.INTERNAL_get_startColor(out color);
                return color;
            }
            set
            {
                this.INTERNAL_set_startColor(ref value);
            }
        }

        public float startDelay { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessI
set; }

        public float startLifetime { [MethodImpl(                    alC
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.                    essI
set; }

```

```
        public float startRotation { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float startSize { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float startSpeed { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float time { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        [StructLayout(LayoutKind.Sequential)]
        public struct CollisionEvent
        {
            private Vector3 m_Intersection;
            private Vector3 m_Normal;
            private Vector3 m_Velocity;
            private int m_ColliderInstanceID;
            public Vector3 intersection
            {
                get
                {
                    return this.m_Intersection;
                }
            }
            public Vector3 normal
            {
                get
                {
                    return this.m_Normal;
                }
            }
            public Vector3 velocity
            {
                get
                {
                    return this.m_Velocity;
                }
            }
            public Collider collider
            {
                get
                {
                    return ParticleSystem.InstanceIDToCollider(this.m_ColliderInstanceID);
                }
            }
        }

        [StructLayout(LayoutKind.Sequential)]
        public struct Particle
        {
            private Vector3 m_Position;
            private Vector3 m_Velocity;
            private Vector3 m_AnimatedVelocity;
            private Vector3 m_AxisOfRotation;
            private float m_Rotation;
            private float m_AngularVelocity;
            private float m_Size;
            private Color32 m_Color;
            private uint m_RandomSeed;
            private float m_Lifetime;
            private float m_StartLifetime;
            private float m_EmitAccumulator0;
            private float m_EmitAccumulator1;
            public Vector3 position
            {
                get
                {
                    return this.m_Position;
                }
                set
                {
                    this.m_Position = value;
                }
            }
            public Vector3 velocity
            {
                get
                {
                    return this.m_Velocity;
                }
                set
                {
                    this.m_Velocity = value;
                }
            }
            public float lifetime
            {
                get
                {
                    return this.m_Lifetime;
                }
                set
                {
                    this.m_Lifetime = value;
                }
            }
            public float startLifetime
            {
                get
                {
                    return this.m_StartLifetime;
                }
                set
                {
                    this.m_StartLifetime = value;
                }
            }
            public float size
            {
```

推荐: 1

```
            get
            {
                return this.m_Size;
            }
            set
            {
                this.m_Size = value;
            }
        }
        public Vector3 axisOfRotation
        {
            get
            {
                return this.m_AxisOfRotation;
            }
            set
            {
                this.m_AxisOfRotation = value;
            }
        }
        public float rotation
        {
            get
            {
                return (this.m_Rotation * 57.29578f);
            }
            set
            {
                this.m_Rotation = value * 0.01745329f;
            }
        }
        public float angularVelocity
        {
            get
            {
                return (this.m_AngularVelocity * 57.29578f);
            }
            set
            {
                this.m_AngularVelocity = value * 0.01745329f;
            }
        }
        public Color32 color
        {
            get
            {
                return this.m_Color;
            }
            set
            {
                this.m_Color = value;
            }
        }
        [Obsolete("randomValue property is deprecated. Use randomSeed in
control random behavior of particles.")]
        public float randomValue
        {
            get
```

推荐: 1

```
                {
                    return
BitConverter.ToSingle(BitConverter.GetBytes(this.m_RandomSeed), 0);
                }
                set
                {
                    this.m_RandomSeed =
BitConverter.ToUInt32(BitConverter.GetBytes(value), 0);
                }
            }
            public uint randomSeed
            {
                get
                {
                    return this.m_RandomSeed;
                }
                set
                {
                    this.m_RandomSeed = value;
                }
            }
        }
    }
}
```

回到顶部(go to top)

## UnityEngine.Behaviour

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;

    public class Behaviour : Component
    {
        public bool enabled { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(g o)

## UnityEngine.Collider

推荐: 1

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;

    public class Collider : Component
    {
        public Vector3 ClosestPointOnBounds(Vector3 position)
        {
            return INTERNAL_CALL_ClosestPointOnBounds(this, ref position);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern Vector3 INTERNAL_CALL_ClosestPointOnBounds(Collider self, ref Vector3 position);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern bool INTERNAL_CALL_Internal_Raycast(Collider col, ref Ray ray, out RaycastHit hitInfo, float distance);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_bounds(out Bounds value);
        private static bool Internal_Raycast(Collider col, Ray ray, out RaycastHit hitInfo, float distance)
        {
            return INTERNAL_CALL_Internal_Raycast(col, ref ray, out hitInfo, distance);
        }

        public bool Raycast(Ray ray, out RaycastHit hitInfo, float distance)
        {
            return Internal_Raycast(this, ray, out hitInfo, distance);
        }

        public Rigidbody attachedRigidbody { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Bounds bounds
        {
            get
            {
                Bounds bounds;
                this.INTERNAL_get_bounds(out bounds);
                return bounds;
            }
        }

        public bool enabled { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool isTrigger { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessI
set; }

        public PhysicMaterial material { [MethodIm                    tern
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.              essI
set; }
```

```

        public PhysicMaterial sharedMaterial {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(go to top)

## UnityEngine.Rigidbody

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;
    using UnityEngine.Internal;

    public sealed class Rigidbody : Component
    {
        [ExcludeFromDocs]
        public void AddExplosionForce(float explosionForce, Vector3
explosionPosition, float explosionRadius)
        {
            ForceMode force = ForceMode.Force;
            float upwardsModifier = 0f;
            INTERNAL_CALL_AddExplosionForce(this, explosionForce, ref
explosionPosition, explosionRadius, upwardsModifier, force);
        }

        [ExcludeFromDocs]
        public void AddExplosionForce(float explosionForce, Vector3
explosionPosition, float explosionRadius, float upwardsModifier)
        {
            ForceMode force = ForceMode.Force;
            INTERNAL_CALL_AddExplosionForce(this, explosionForce, ref
explosionPosition, explosionRadius, upwardsModifier, force);
        }

        public void AddExplosionForce(float explosionForce, Vector3
explosionPosition, float explosionRadius, [DefaultValue("0.0F")] float upwardsModifier,
[DefaultValue("ForceMode.Force")] ForceMode mode)
        {
            INTERNAL_CALL_AddExplosionForce(this, explosionForce, ref
explosionPosition, explosionRadius, upwardsModifier, mode);
        }

        [ExcludeFromDocs]
        public void AddForce(Vector3 force)
        {
            ForceMode mode = ForceMode.Force;
            INTERNAL_CALL_AddForce(this, ref force, mode);
```

推荐: 1

```
        }

        public void AddForce(Vector3 force, [DefaultValue("ForceMode.Force")]
ForceMode mode)
        {
            INTERNAL_CALL_AddForce(this, ref force, mode);
        }

        [ExcludeFromDocs]
        public void AddForce(float x, float y, float z)
        {
            ForceMode force = ForceMode.Force;
            this.AddForce(x, y, z, force);
        }

        public void AddForce(float x, float y, float z,
[DefaultValue("ForceMode.Force")] ForceMode mode)
        {
            this.AddForce(new Vector3(x, y, z), mode);
        }

        [ExcludeFromDocs]
        public void AddForceAtPosition(Vector3 force, Vector3 position)
        {
            ForceMode mode = ForceMode.Force;
            INTERNAL_CALL_AddForceAtPosition(this, ref force, ref position, mode);
        }

        public void AddForceAtPosition(Vector3 force, Vector3 position,
[DefaultValue("ForceMode.Force")] ForceMode mode)
        {
            INTERNAL_CALL_AddForceAtPosition(this, ref force, ref position, mode);
        }

        [ExcludeFromDocs]
        public void AddRelativeForce(Vector3 force)
        {
            ForceMode mode = ForceMode.Force;
            INTERNAL_CALL_AddRelativeForce(this, ref force, mode);
        }

        public void AddRelativeForce(Vector3 force,
[DefaultValue("ForceMode.Force")] ForceMode mode)
        {
            INTERNAL_CALL_AddRelativeForce(this, ref force, mode);
        }

        [ExcludeFromDocs]
        public void AddRelativeForce(float x, float y, float z)
        {
            ForceMode force = ForceMode.Force;
            this.AddRelativeForce(x, y, z, force);
        }

        public void AddRelativeForce(float x, flo
[DefaultValue("ForceMode.Force")] ForceMode mode)
        {
            this.AddRelativeForce(new Vector3(x, y, z), mode);
```

推荐: 1

```
        }

        [ExcludeFromDocs]
        public void AddRelativeTorque(Vector3 torque)
        {
            ForceMode force = ForceMode.Force;
            INTERNAL_CALL_AddRelativeTorque(this, ref torque, force);
        }

        public void AddRelativeTorque(Vector3 torque,
[DefaultValue("ForceMode.Force")] ForceMode mode)
        {
            INTERNAL_CALL_AddRelativeTorque(this, ref torque, mode);
        }

        [ExcludeFromDocs]
        public void AddRelativeTorque(float x, float y, float z)
        {
            ForceMode force = ForceMode.Force;
            this.AddRelativeTorque(x, y, z, force);
        }

        public void AddRelativeTorque(float x, float y, float z,
[DefaultValue("ForceMode.Force")] ForceMode mode)
        {
            this.AddRelativeTorque(new Vector3(x, y, z), mode);
        }

        [ExcludeFromDocs]
        public void AddTorque(Vector3 torque)
        {
            ForceMode force = ForceMode.Force;
            INTERNAL_CALL_AddTorque(this, ref torque, force);
        }

        public void AddTorque(Vector3 torque, [DefaultValue("ForceMode.Force")]
ForceMode mode)
        {
            INTERNAL_CALL_AddTorque(this, ref torque, mode);
        }

        [ExcludeFromDocs]
        public void AddTorque(float x, float y, float z)
        {
            ForceMode force = ForceMode.Force;
            this.AddTorque(x, y, z, force);
        }

        public void AddTorque(float x, float y, float z,
[DefaultValue("ForceMode.Force")] ForceMode mode)
        {
            this.AddTorque(new Vector3(x, y, z), mode);
        }

        public Vector3 ClosestPointOnBounds(Vecto
        {
            return INTERNAL_CALL_ClosestPointOnBo                         ;
        }

        public Vector3 GetPointVelocity(Vector3 worldPoint)
        {
            return INTERNAL_CALL_GetPointVelocity(this, ref worldPoint);
        }

        public Vector3 GetRelativePointVelocity(Vector3 relativePoint)
        {
            return INTERNAL_CALL_GetRelativePointVelocity(this, ref relativePoint);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_AddExplosionForce(Rigidbody self, float explosionForce, ref Vector3 explosionPosition, float explosionRadius, float upwardsModifier, ForceMode mode);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_AddForce(Rigidbody self, ref Vector3 force, ForceMode mode);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_AddForceAtPosition(Rigidbody self, ref Vector3 force, ref Vector3 position, ForceMode mode);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_AddRelativeForce(Rigidbody self, ref Vector3 force, ForceMode mode);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_AddRelativeTorque(Rigidbody self, ref Vector3 torque, ForceMode mode);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_AddTorque(Rigidbody self, ref Vector3 torque, ForceMode mode);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern Vector3 INTERNAL_CALL_ClosestPointOnBounds(Rigidbody self, ref Vector3 position);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern Vector3 INTERNAL_CALL_GetPointVelocity(Rigidbody self, ref Vector3 worldPoint);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern Vector3 INTERNAL_CALL_GetRelativePointVelocity(Rigidbody self, ref Vector3 relativePoint);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern bool INTERNAL_CALL_IsSleeping(Rigidbody self);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_MovePosition(Rigidbody self, ref Vector3 position);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_MoveRotation(Rigidbody self, ref Quaternion rot);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_SetDensity(Rigidbody self, float density);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_Sleep(Rigidbody self);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern bool INTERNAL_CALL_SweepTest(Rigidbody self, r
Vector3 direction, out RaycastHit hitInfo, float dist
        [MethodImpl(MethodImplOptions.InternalCal
        private static extern RaycastHit[] INTERN                          idb
self, ref Vector3 direction, float distance);
```

```
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_WakeUp(Rigidbody self);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_angularVelocity(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_centerOfMass(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_inertiaTensor(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_inertiaTensorRotation(out Quaternion value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_position(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_rotation(out Quaternion value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_velocity(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_worldCenterOfMass(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_angularVelocity(ref Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_centerOfMass(ref Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_inertiaTensor(ref Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_inertiaTensorRotation(ref Quaternion value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_position(ref Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_rotation(ref Quaternion value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_velocity(ref Vector3 value);
        public bool IsSleeping()
        {
            return INTERNAL_CALL_IsSleeping(this);
        }

        public void MovePosition(Vector3 position)
        {
            INTERNAL_CALL_MovePosition(this, ref position);
        }

        public void MoveRotation(Quaternion rot)
        {
            INTERNAL_CALL_MoveRotation(this, ref rot);
        }

        public void SetDensity(float density)
        {
            INTERNAL_CALL_SetDensity(this, density);
        }

        [Obsolete("use Rigidbody.maxAngularVeloci
        public void SetMaxAngularVelocity(float a
        {
            this.maxAngularVelocity = a;
```

推荐: 1

```
        }

        public void Sleep()
        {
            INTERNAL_CALL_Sleep(this);
        }

        [ExcludeFromDocs]
        public bool SweepTest(Vector3 direction, out RaycastHit hitInfo)
        {
            float positiveInfinity = float.PositiveInfinity;
            return INTERNAL_CALL_SweepTest(this, ref direction, out hitInfo, positiveInfinity);
        }

        public bool SweepTest(Vector3 direction, out RaycastHit hitInfo, [DefaultValue("Mathf.Infinity")] float distance)
        {
            return INTERNAL_CALL_SweepTest(this, ref direction, out hitInfo, distance);
        }

        [ExcludeFromDocs]
        public RaycastHit[] SweepTestAll(Vector3 direction)
        {
            float positiveInfinity = float.PositiveInfinity;
            return INTERNAL_CALL_SweepTestAll(this, ref direction, positiveInfinity);
        }

        public RaycastHit[] SweepTestAll(Vector3 direction, [DefaultValue("Mathf.Infinity")] float distance)
        {
            return INTERNAL_CALL_SweepTestAll(this, ref direction, distance);
        }

        public void WakeUp()
        {
            INTERNAL_CALL_WakeUp(this);
        }

        public float angularDrag { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector3 angularVelocity
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_angularVelocity(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_angularVelocity
            }
        }

        public Vector3 centerOfMass
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_centerOfMass(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_centerOfMass(ref value);
            }
        }

        public CollisionDetectionMode collisionDetectionMode { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public RigidbodyConstraints constraints { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool detectCollisions { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float drag { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool freezeRotation { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector3 inertiaTensor
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_inertiaTensor(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_inertiaTensor(ref value);
            }
        }

        public Quaternion inertiaTensorRotation
        {
            get
            {
                Quaternion quaternion;
                this.INTERNAL_get_inertiaTensorRo
                return quaternion;
            }
            set
```

推荐: 1

```
            {
                this.INTERNAL_set_inertiaTensorRotation(ref value);
            }
        }

        public RigidbodyInterpolation interpolation { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool isKinematic { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float mass { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float maxAngularVelocity { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector3 position
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_position(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_position(ref value);
            }
        }

        public Quaternion rotation
        {
            get
            {
                Quaternion quaternion;
                this.INTERNAL_get_rotation(out quaternion);
                return quaternion;
            }
            set
            {
                this.INTERNAL_set_rotation(ref value);
            }
        }

        public float sleepAngularVelocity { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float sleepVelocity { [MethodImpl(MethodImplOptions.InternalC
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.                    essI
set; }

        public int solverIterationCount {
```

```
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool useConeFriction { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

        public bool useGravity { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

        public Vector3 velocity
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_velocity(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_velocity(ref value);
            }
        }

        public Vector3 worldCenterOfMass
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_worldCenterOfMass(out vector);
                return vector;
            }
        }
    }
}
```

回到顶部(go to top)

## UnityEngine.AudioListener

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;

    public sealed class AudioListener : Behaviour
    {
        [Obsolete("GetOutputData returning a float
and pass a pre allocated array instead.")]
        public static float[] GetOutputData(int nu
        {
```

推荐: 1

```
            float[] samples = new float[numSamples];
            GetOutputDataHelper(samples, channel);
            return samples;
        }

        public static void GetOutputData(float[] samples, int channel)
        {
            GetOutputDataHelper(samples, channel);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void GetOutputDataHelper(float[] samples, int channel);
        [Obsolete("GetSpectrumData returning a float[] is deprecated, use GetOutputData and pass a pre allocated array instead.")]
        public static float[] GetSpectrumData(int numSamples, int channel, FFTWindow window)
        {
            float[] samples = new float[numSamples];
            GetSpectrumDataHelper(samples, channel, window);
            return samples;
        }

        public static void GetSpectrumData(float[] samples, int channel, FFTWindow window)
        {
            GetSpectrumDataHelper(samples, channel, window);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void GetSpectrumDataHelper(float[] samples, int channel, FFTWindow window);

        public static bool pause { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public AudioVelocityUpdateMode velocityUpdateMode { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public static float volume { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(go to top)

## UnityEngine.Camera

推荐: 1

```
namespace UnityEngine
{
```

```
using System;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using UnityEngine.Internal;

public sealed class Camera : Behaviour
{
    public Matrix4x4 CalculateObliqueMatrix(Vector4 clipPlane)
    {
        return INTERNAL_CALL_CalculateObliqueMatrix(this, ref clipPlane);
    }

    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public extern void CopyFrom(Camera other);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall, Obsolete("Camera.DoClear is deprecated and may be removed in the future.")]
    public extern void DoClear();
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    public static extern int GetAllCameras(Camera[] cameras);
    [MethodImpl(MethodImplOptions.InternalCall), Obsolete("use Screen.height instead."), WrapperlessIcall]
    public extern float GetScreenHeight();
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall, Obsolete("use Screen.width instead.")]
    public extern float GetScreenWidth();
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Matrix4x4 INTERNAL_CALL_CalculateObliqueMatrix(Camera self, ref Vector4 clipPlane);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void INTERNAL_CALL_ResetAspect(Camera self);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void INTERNAL_CALL_ResetProjectionMatrix(Camera self);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void INTERNAL_CALL_ResetReplacementShader(Camera self);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern void INTERNAL_CALL_ResetWorldToCameraMatrix(Camera self);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Ray INTERNAL_CALL_ScreenPointToRay(Camera self, ref Vector3 position);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Vector3 INTERNAL_CALL_ScreenToViewportPoint(Camera self, ref Vector3 position);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Vector3 INTERNAL_CALL_ScreenToWorldPoint(Camera self, ref Vector3 position);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Ray INTERNAL_CALL_ViewportPointToRay(Camera self, ref Vector3 position);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Vector3 INTERNAL_CALL_ViewportToScreenPoint(Ca
self, ref Vector3 position);
    [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
    private static extern Vector3 INTERNAL_CA                    (Cam
self, ref Vector3 position);
    [MethodImpl(MethodImplOptions.InternalCal
    private static extern Vector3 INTERNAL_CALL_WorldToScreenPoint(Camera self,
```

推荐: 1

```
ref Vector3 position);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern Vector3 INTERNAL_CALL_WorldToViewportPoint(Camera self, ref Vector3 position);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_backgroundColor(out Color value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_cameraToWorldMatrix(out Matrix4x4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_pixelRect(out Rect value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_projectionMatrix(out Matrix4x4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_rect(out Rect value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_velocity(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_worldToCameraMatrix(out Matrix4x4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool Internal_RenderToCubemapRT(RenderTexture cubemap, int faceMask);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool Internal_RenderToCubemapTexture(Cubemap cubemap, int faceMask);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_backgroundColor(ref Color value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_pixelRect(ref Rect value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_projectionMatrix(ref Matrix4x4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_rect(ref Rect value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_worldToCameraMatrix(ref Matrix4x4 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern bool IsFiltered(GameObject go);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Render();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void RenderDontRestore();
        [ExcludeFromDocs]
        public bool RenderToCubemap(Cubemap cubemap)
        {
            int faceMask = 0x3f;
            return this.RenderToCubemap(cubemap, faceMask);
        }

        [ExcludeFromDocs]
        public bool RenderToCubemap(RenderTexture cubemap)
        {
            int faceMask = 0x3f;
            return this.RenderToCubemap(cubemap, faceMask);
        }

        public bool RenderToCubemap(Cubemap cubem                in
faceMask)
        {
            return this.Internal_RenderToCubemapTexture(cubemap, faceMask);
```

推荐: 1

```
        }

        public bool RenderToCubemap(RenderTexture cubemap, [DefaultValue("63")] int faceMask)
        {
            return this.Internal_RenderToCubemapRT(cubemap, faceMask);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void RenderWithShader(Shader shader, string replacementTag);
        public void ResetAspect()
        {
            INTERNAL_CALL_ResetAspect(this);
        }

        public void ResetProjectionMatrix()
        {
            INTERNAL_CALL_ResetProjectionMatrix(this);
        }

        public void ResetReplacementShader()
        {
            INTERNAL_CALL_ResetReplacementShader(this);
        }

        public void ResetWorldToCameraMatrix()
        {
            INTERNAL_CALL_ResetWorldToCameraMatrix(this);
        }

        public Ray ScreenPointToRay(Vector3 position)
        {
            return INTERNAL_CALL_ScreenPointToRay(this, ref position);
        }

        public Vector3 ScreenToViewportPoint(Vector3 position)
        {
            return INTERNAL_CALL_ScreenToViewportPoint(this, ref position);
        }

        public Vector3 ScreenToWorldPoint(Vector3 position)
        {
            return INTERNAL_CALL_ScreenToWorldPoint(this, ref position);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetReplacementShader(Shader shader, string replacementTag);
        public void SetTargetBuffers(RenderBuffer colorBuffer, RenderBuffer depthBuffer)
        {
            this.SetTargetBuffersImpl(out colorBuffer, out depthBuffer);
        }

        public void SetTargetBuffers(RenderBuffer                          uffe
depthBuffer)
        {
            this.SetTargetBuffersMRTImpl(colorBuf
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetTargetBuffersImpl(out RenderBuffer color, out
RenderBuffer depth);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetTargetBuffersMRTImpl(RenderBuffer[] color, out
RenderBuffer depth);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public static extern void SetupCurrent(Camera cur);
        public Ray ViewportPointToRay(Vector3 position)
        {
            return INTERNAL_CALL_ViewportPointToRay(this, ref position);
        }

        public Vector3 ViewportToScreenPoint(Vector3 position)
        {
            return INTERNAL_CALL_ViewportToScreenPoint(this, ref position);
        }

        public Vector3 ViewportToWorldPoint(Vector3 position)
        {
            return INTERNAL_CALL_ViewportToWorldPoint(this, ref position);
        }

        public Vector3 WorldToScreenPoint(Vector3 position)
        {
            return INTERNAL_CALL_WorldToScreenPoint(this, ref position);
        }

        public Vector3 WorldToViewportPoint(Vector3 position)
        {
            return INTERNAL_CALL_WorldToViewportPoint(this, ref position);
        }

        public RenderingPath actualRenderingPath {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public static Camera[] allCameras {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public static int allCamerasCount {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public float aspect { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

        public Color backgroundColor
        {
            get
            {
                Color color;
                this.INTERNAL_get_backgroundColor(out color);
                return color;
            }
            set
            {
```

推荐: 1

```
                    this.INTERNAL_set_backgroundColor(ref value);
                }
            }

            public Matrix4x4 cameraToWorldMatrix
            {
                get
                {
                    Matrix4x4 matrixx;
                    this.INTERNAL_get_cameraToWorldMatrix(out matrixx);
                    return matrixx;
                }
            }

            public CameraClearFlags clearFlags {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

            public bool clearStencilAfterLightingPass {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

            public int cullingMask { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

            public static Camera current { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; }

            public float depth { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

            public DepthTextureMode depthTextureMode {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

            public int eventMask { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

            [Obsolete("use Camera.farClipPlane instead.")]
            public float far
            {
                get
                {
                    return this.farClipPlane;
                }
                set
                {
                    this.farClipPlane = value;
                }
            }

            public float farClipPlane { [MethodImpl(M                          alCa
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.                      essI
set; }

```

推荐: 1

```
        public float fieldOfView { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        [Obsolete("use Camera.fieldOfView instead.")]
        public float fov
        {
            get
            {
                return this.fieldOfView;
            }
            set
            {
                this.fieldOfView = value;
            }
        }

        public bool hdr { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool isOrthoGraphic
        {
            get
            {
                return this.orthographic;
            }
            set
            {
                this.orthographic = value;
            }
        }

        public float[] layerCullDistances { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool layerCullSpherical { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public static Camera main { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        [Obsolete("use Camera.main instead.")]
        public static Camera mainCamera
        {
            get
            {
                return main;
            }
        }

        [Obsolete("use Camera.nearClipPlane instead.")]
        public float near
        {
            get
            {
```

推荐: 1

```
                    return this.nearClipPlane;
                }
                set
                {
                    this.nearClipPlane = value;
                }
            }

            public float nearClipPlane { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

            public bool orthographic { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

            public float orthographicSize { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

            public float pixelHeight { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

            public Rect pixelRect
            {
                get
                {
                    Rect rect;
                    this.INTERNAL_get_pixelRect(out rect);
                    return rect;
                }
                set
                {
                    this.INTERNAL_set_pixelRect(ref value);
                }
            }

            public float pixelWidth { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

            public Matrix4x4 projectionMatrix
            {
                get
                {
                    Matrix4x4 matrixx;
                    this.INTERNAL_get_projectionMatrix(out matrixx);
                    return matrixx;
                }
                set
                {
                    this.INTERNAL_set_projectionMatrix(ref value);
                }
            }

            public Rect rect
            {
                get
                {
```

```
                Rect rect;
                this.INTERNAL_get_rect(out rect);
                return rect;
            }
            set
            {
                this.INTERNAL_set_rect(ref value);
            }
        }

        public RenderingPath renderingPath { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float stereoConvergence { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool stereoEnabled { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public float stereoSeparation { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public RenderTexture targetTexture { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public TransparencySortMode transparencySortMode { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool useOcclusionCulling { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector3 velocity
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_velocity(out vector);
                return vector;
            }
        }

        public Matrix4x4 worldToCameraMatrix
        {
            get
            {
                Matrix4x4 matrixx;
                this.INTERNAL_get_worldToCameraMatrix(out matrixx);
                return matrixx;
            }
            set
            {
                this.INTERNAL_set_worldToCameraMatrix(ref value);
            }
        }
    }
}
```

[回到顶部(go to top)](#)

## UnityEngine.Animator

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;
    using UnityEngine.Internal;

    public sealed class Animator : Behaviour
    {
        [ExcludeFromDocs]
        public void CrossFade(int stateNameHash, float transitionDuration)
        {
            float negativeInfinity = float.NegativeInfinity;
            int layer = -1;
            this.CrossFade(stateNameHash, transitionDuration, layer, negativeInfinity);
        }

        [ExcludeFromDocs]
        public void CrossFade(string stateName, float transitionDuration)
        {
            float negativeInfinity = float.NegativeInfinity;
            int layer = -1;
            this.CrossFade(stateName, transitionDuration, layer, negativeInfinity);
        }

        [ExcludeFromDocs]
        public void CrossFade(int stateNameHash, float transitionDuration, int layer)
        {
            float negativeInfinity = float.NegativeInfinity;
            this.CrossFade(stateNameHash, transitionDuration, layer, negativeInfinity);
        }

        [ExcludeFromDocs]
        public void CrossFade(string stateName, float transitionDuration, in
        {
            float negativeInfinity = float.NegativeInfinity;
            this.CrossFade(stateName, transitionD                        eInf
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
```

推荐: 1

```
        public extern void CrossFade(int stateNameHash, float transitionDuration, [DefaultValue("-1")] int layer, [DefaultValue("float.NegativeInfinity")] float normalizedTime);
        public void CrossFade(string stateName, float transitionDuration, [DefaultValue("-1")] int layer, [DefaultValue("float.NegativeInfinity")] float normalizedTime)
        {
            this.CrossFade(StringToHash(stateName), transitionDuration, layer, normalizedTime);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern void EvaluateSM();
        [Obsolete("ForceStateNormalizedTime is deprecated. Please use Play or CrossFade instead.")]
        public void ForceStateNormalizedTime(float normalizedTime)
        {
            this.Play(0, 0, normalizedTime);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern AnimatorTransitionInfo GetAnimatorTransitionInfo(int layerIndex);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern Transform GetBoneTransform(HumanBodyBones humanBoneId);
        public bool GetBool(int id)
        {
            return this.GetBoolID(id);
        }

        public bool GetBool(string name)
        {
            return this.GetBoolString(name);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool GetBoolID(int id);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool GetBoolString(string name);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern AnimationInfo[] GetCurrentAnimationClipState(int layerIndex);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern AnimatorStateInfo GetCurrentAnimatorStateInfo(int layerIndex);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern string GetCurrentStateName(int layerIndex);
        public float GetFloat(int id)
        {
            return this.GetFloatID(id);
        }

        public float GetFloat(string name)
        {
            return this.GetFloatString(name);
        }

        [MethodImpl(MethodImplOptions.InternalCal
        private extern float GetFloatID(int id);
        [MethodImpl(MethodImplOptions.InternalCal
```

推荐: 1

```
        private extern float GetFloatString(string name);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern Vector3 GetIKPosition(AvatarIKGoal goal);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern float GetIKPositionWeight(AvatarIKGoal goal);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern Quaternion GetIKRotation(AvatarIKGoal goal);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern float GetIKRotationWeight(AvatarIKGoal goal);
        public int GetInteger(int id)
        {
            return this.GetIntegerID(id);
        }

        public int GetInteger(string name)
        {
            return this.GetIntegerString(name);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern int GetIntegerID(int id);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern int GetIntegerString(string name);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern string GetLayerName(int layerIndex);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern float GetLayerWeight(int layerIndex);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern AnimationInfo[] GetNextAnimationClipState(int layerIndex);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern AnimatorStateInfo GetNextAnimatorStateInfo(int layerIndex);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern string GetNextStateName(int layerIndex);
        [Obsolete("GetQuaternion is deprecated.")]
        public Quaternion GetQuaternion(int id)
        {
            return Quaternion.identity;
        }

        [Obsolete("GetQuaternion is deprecated.")]
        public Quaternion GetQuaternion(string name)
        {
            return Quaternion.identity;
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern string GetStats();
        [Obsolete("GetVector is deprecated.")]
        public Vector3 GetVector(int id)
        {
            return Vector3.zero;
        }

        [Obsolete("GetVector is deprecated.")]
        public Vector3 GetVector(string name)
        {
            return Vector3.zero;
        }
```

推荐: 1

```

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_MatchTarget(Animator self, ref Vector3 matchPosition, ref Quaternion matchRotation, AvatarTarget targetBodyPart, ref MatchTargetWeightMask weightMask, float startNormalizedTime, float targetNormalizedTime);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_SetIKPosition(Animator self, AvatarIKGoal goal, ref Vector3 goalPosition);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_SetIKRotation(Animator self, AvatarIKGoal goal, ref Quaternion goalRotation);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_SetLookAtPosition(Animator self, ref Vector3 lookAtPosition);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_bodyPosition(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_bodyRotation(out Quaternion value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_rootPosition(out Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_rootRotation(out Quaternion value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_bodyPosition(ref Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_bodyRotation(ref Quaternion value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_rootPosition(ref Vector3 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_rootRotation(ref Quaternion value);
        [ExcludeFromDocs]
        public void InterruptMatchTarget()
        {
            bool completeMatch = true;
            this.InterruptMatchTarget(completeMatch);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void InterruptMatchTarget([DefaultValue("true")] bool completeMatch);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern bool IsBoneTransform(Transform transform);
        [MethodImpl(MethodImplOptions.InternalCall), Obsolete("use mask and layers to control subset of transfroms in a skeleton", true), WrapperlessIcall]
        public extern bool IsControlled(Transform transform);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern bool IsInTransition(int layerIndex);
        public bool IsParameterControlledByCurve(int id)
        {
            return this.IsParameterControlledByCurveID(id);
        }

        public bool IsParameterControlledByCurve(string name)
        {
            return this.IsParameterControlledByCu
        }

```

推荐: 1

```
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool IsParameterControlledByCurveID(int id);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool IsParameterControlledByCurveString(string name);
        [ExcludeFromDocs]
        public void MatchTarget(Vector3 matchPosition, Quaternion matchRotation, AvatarTarget targetBodyPart, MatchTargetWeightMask weightMask, float startNormalizedTime)
        {
            float targetNormalizedTime = 1f;
            INTERNAL_CALL_MatchTarget(this, ref matchPosition, ref matchRotation, targetBodyPart, ref weightMask, startNormalizedTime, targetNormalizedTime);
        }

        public void MatchTarget(Vector3 matchPosition, Quaternion matchRotation, AvatarTarget targetBodyPart, MatchTargetWeightMask weightMask, float startNormalizedTime, [DefaultValue("1")] float targetNormalizedTime)
        {
            INTERNAL_CALL_MatchTarget(this, ref matchPosition, ref matchRotation, targetBodyPart, ref weightMask, startNormalizedTime, targetNormalizedTime);
        }

        [ExcludeFromDocs]
        public void Play(int stateNameHash)
        {
            float negativeInfinity = float.NegativeInfinity;
            int layer = -1;
            this.Play(stateNameHash, layer, negativeInfinity);
        }

        [ExcludeFromDocs]
        public void Play(string stateName)
        {
            float negativeInfinity = float.NegativeInfinity;
            int layer = -1;
            this.Play(stateName, layer, negativeInfinity);
        }

        [ExcludeFromDocs]
        public void Play(int stateNameHash, int layer)
        {
            float negativeInfinity = float.NegativeInfinity;
            this.Play(stateNameHash, layer, negativeInfinity);
        }

        [ExcludeFromDocs]
        public void Play(string stateName, int layer)
        {
            float negativeInfinity = float.NegativeInfinity;
            this.Play(stateName, layer, negativeInfinity);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Play(int stateNameHash, [DefaultValue("-1")] int [DefaultValue("float.NegativeInfinity")] float normal
        public void Play(string stateName, [Defau                                  f, [DefaultValue("float.NegativeInfinity")] float normal
        {
```

推荐: 1

```
            this.Play(StringToHash(stateName), layer, normalizedTime);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Rebind();
        public void ResetTrigger(int id)
        {
            this.ResetTriggerID(id);
        }

        public void ResetTrigger(string name)
        {
            this.ResetTriggerString(name);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void ResetTriggerID(int id);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void ResetTriggerString(string name);
        public void SetBool(int id, bool value)
        {
            this.SetBoolID(id, value);
        }

        public void SetBool(string name, bool value)
        {
            this.SetBoolString(name, value);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetBoolID(int id, bool value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetBoolString(string name, bool value);
        public void SetFloat(int id, float value)
        {
            this.SetFloatID(id, value);
        }

        public void SetFloat(string name, float value)
        {
            this.SetFloatString(name, value);
        }

        public void SetFloat(int id, float value, float dampTime, float deltaTime)
        {
            this.SetFloatIDDamp(id, value, dampTime, deltaTime);
        }

        public void SetFloat(string name, float value, float dampTime, float deltaTime)
        {
            this.SetFloatStringDamp(name, value, dampTime, deltaTime);
        }

        [MethodImpl(MethodImplOptions.InternalCal
        private extern void SetFloatID(int id, fl
        [MethodImpl(MethodImplOptions.InternalCal
        private extern void SetFloatIDDamp(int id, float value, float dampTime,
float deltaTime);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetFloatString(string name, float value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetFloatStringDamp(string name, float value, float
dampTime, float deltaTime);
        public void SetIKPosition(AvatarIKGoal goal, Vector3 goalPosition)
        {
            INTERNAL_CALL_SetIKPosition(this, goal, ref goalPosition);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetIKPositionWeight(AvatarIKGoal goal, float value);
        public void SetIKRotation(AvatarIKGoal goal, Quaternion goalRotation)
        {
            INTERNAL_CALL_SetIKRotation(this, goal, ref goalRotation);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetIKRotationWeight(AvatarIKGoal goal, float value);
        public void SetInteger(int id, int value)
        {
            this.SetIntegerID(id, value);
        }

        public void SetInteger(string name, int value)
        {
            this.SetIntegerString(name, value);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetIntegerID(int id, int value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetIntegerString(string name, int value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetLayerWeight(int layerIndex, float weight);
        public void SetLookAtPosition(Vector3 lookAtPosition)
        {
            INTERNAL_CALL_SetLookAtPosition(this, ref lookAtPosition);
        }

        [ExcludeFromDocs]
        public void SetLookAtWeight(float weight)
        {
            float clampWeight = 0.5f;
            float eyesWeight = 0f;
            float headWeight = 1f;
            float bodyWeight = 0f;
            this.SetLookAtWeight(weight, bodyWeight, headWeight, eyesWeight,
clampWeight);
        }

        [ExcludeFromDocs]
        public void SetLookAtWeight(float weight, float bodyWeight)
        {
            float clampWeight = 0.5f;
            float eyesWeight = 0f;
            float headWeight = 1f;
```

推荐: 1

```
            this.SetLookAtWeight(weight, bodyWeight, headWeight, eyesWeight, clampWeight);
        }

        [ExcludeFromDocs]
        public void SetLookAtWeight(float weight, float bodyWeight, float headWeight)
        {
            float clampWeight = 0.5f;
            float eyesWeight = 0f;
            this.SetLookAtWeight(weight, bodyWeight, headWeight, eyesWeight, clampWeight);
        }

        [ExcludeFromDocs]
        public void SetLookAtWeight(float weight, float bodyWeight, float headWeight, float eyesWeight)
        {
            float clampWeight = 0.5f;
            this.SetLookAtWeight(weight, bodyWeight, headWeight, eyesWeight, clampWeight);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetLookAtWeight(float weight, [DefaultValue("0.00f")] float bodyWeight, [DefaultValue("1.00f")] float headWeight, [DefaultValue("0.00f")] float eyesWeight, [DefaultValue("0.50f")] float clampWeight);
        [Obsolete("SetQuaternion is deprecated.")]
        public void SetQuaternion(int id, Quaternion value)
        {
        }

        [Obsolete("SetQuaternion is deprecated.")]
        public void SetQuaternion(string name, Quaternion value)
        {
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetTarget(AvatarTarget targetIndex, float targetNormalizedTime);
        public void SetTrigger(int id)
        {
            this.SetTriggerID(id);
        }

        public void SetTrigger(string name)
        {
            this.SetTriggerString(name);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetTriggerID(int id);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void SetTriggerString(string name);
        [Obsolete("SetVector is deprecated.")]
        public void SetVector(int id, Vector3 val
        {
        }
        [Obsolete("SetVector is deprecated.")]
        public void SetVector(string name, Vector3 value)
        {
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void StartPlayback();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void StartRecording(int frameCount);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void StopPlayback();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void StopRecording();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public static extern int StringToHash(string name);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Update(float deltaTime);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern void WriteDefaultPose();

        internal bool allowConstantClipSamplingOptimization { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        [Obsolete("Use AnimationMode.updateMode instead")]
        public bool animatePhysics
        {
            get
            {
                return (this.updateMode == AnimatorUpdateMode.AnimatePhysics);
            }
            set
            {
                this.updateMode = !value ? AnimatorUpdateMode.Normal : AnimatorUpdateMode.AnimatePhysics;
            }
        }

        public bool applyRootMotion { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Avatar avatar { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        internal Transform avatarRoot { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Vector3 bodyPosition
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_bodyPosition(ou
                return vector;
            }
```

推荐: 1

```
            set
            {
                this.INTERNAL_set_bodyPosition(ref value);
            }
        }

        public Quaternion bodyRotation
        {
            get
            {
                Quaternion quaternion;
                this.INTERNAL_get_bodyRotation(out quaternion);
                return quaternion;
            }
            set
            {
                this.INTERNAL_set_bodyRotation(ref value);
            }
        }

        public AnimatorCullingMode cullingMode { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector3 deltaPosition { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Quaternion deltaRotation { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public float feetPivotActive { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool fireEvents { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float gravityWeight { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool hasRootMotion { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool hasTransformHierarchy { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public float humanScale { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool isHuman { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        private bool isInManagerList { [MethodImpl(MethodImplOptions.Interna
WrapperlessIcall] get; }

        public bool isMatchingTarget { [MethodImp          erna
WrapperlessIcall] get; }
```

```

        public bool isOptimizable { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public int layerCount { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool layersAffectMassCenter { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float leftFeetBottomHeight { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool logWarnings { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Vector3 pivotPosition { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public float pivotWeight { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public float playbackTime { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float recorderStartTime { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float recorderStopTime { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float rightFeetBottomHeight { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Vector3 rootPosition
        {
            get
            {
                Vector3 vector;
                this.INTERNAL_get_rootPosition(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_rootPosition(ref value);
            }
        }

        public Quaternion rootRotation
        {
            get
            {
                Quaternion quaternion;
```

推荐: 1

```
                this.INTERNAL_get_rootRotation(out quaternion);
                return quaternion;
            }
            set
            {
                this.INTERNAL_set_rootRotation(ref value);
            }
        }

        public RuntimeAnimatorController runtimeAnimatorController { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float speed { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool stabilizeFeet { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        internal bool supportsOnAnimatorMove { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Vector3 targetPosition { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public Quaternion targetRotation { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public AnimatorUpdateMode updateMode { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(go to top)

## UnityEngine.AudioSource

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
    using UnityEngine.Internal;

    public sealed class AudioSource : Behaviour
    {
        [Obsolete("GetOutputData return a float[]                    Outp
passing a pre allocated array instead.")]
        public float[] GetOutputData(int numSampl
        {
            float[] samples = new float[numSamples];
            this.GetOutputDataHelper(samples, channel);
            return samples;
        }

        public void GetOutputData(float[] samples, int channel)
        {
            this.GetOutputDataHelper(samples, channel);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void GetOutputDataHelper(float[] samples, int channel);
        [Obsolete("GetSpectrumData returning a float[] is deprecated, use GetSpectrumData passing a pre allocated array instead.")]
        public float[] GetSpectrumData(int numSamples, int channel, FFTWindow window)
        {
            float[] samples = new float[numSamples];
            this.GetSpectrumDataHelper(samples, channel, window);
            return samples;
        }

        public void GetSpectrumData(float[] samples, int channel, FFTWindow window)
        {
            this.GetSpectrumDataHelper(samples, channel, window);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void GetSpectrumDataHelper(float[] samples, int channel, FFTWindow window);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_Pause(AudioSource self);
        public void Pause()
        {
            INTERNAL_CALL_Pause(this);
        }

        [ExcludeFromDocs]
        public void Play()
        {
            ulong delay = 0L;
            this.Play(delay);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Play([DefaultValue("0")] ulong delay);
        [ExcludeFromDocs]
        public static void PlayClipAtPoint(AudioClip clip, Vector3 position)
        {
            float volume = 1f;
            PlayClipAtPoint(clip, position, volume);
        }

        public static void PlayClipAtPoint(AudioClip clip, Vector3 position, [DefaultValue("1.0F")] float volume)
        {
            GameObject obj2 = new GameObject("One
                transform = { position = position
            };
            AudioSource source = (AudioSource) obj2.AddComponent(typeof(AudioSource));
            source.clip = clip;
            source.volume = volume;
            source.Play();
            UnityEngine.Object.Destroy(obj2, clip.length * Time.timeScale);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void PlayDelayed(float delay);
        [ExcludeFromDocs]
        public void PlayOneShot(AudioClip clip)
        {
            float volumeScale = 1f;
            this.PlayOneShot(clip, volumeScale);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void PlayOneShot(AudioClip clip, [DefaultValue("1.0F")] float volumeScale);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void PlayScheduled(double time);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetScheduledEndTime(double time);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void SetScheduledStartTime(double time);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Stop();

        public bool bypassEffects { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool bypassListenerEffects { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool bypassReverbZones { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public AudioClip clip { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float dopplerLevel { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool ignoreListenerPause { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool ignoreListenerVolume {
[MethodImpl(MethodImplOptions.InternalCall), Wrapper
[MethodImpl(MethodImplOptions.InternalCall), Wrapper

```

```
        public bool isPlaying { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public bool loop { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float maxDistance { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        [Obsolete("maxVolume is not supported anymore. Use min-, maxDistance and rolloffMode instead.", true)]
        public float maxVolume { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float minDistance { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        [Obsolete("minVolume is not supported anymore. Use min-, maxDistance and rolloffMode instead.", true)]
        public float minVolume { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool mute { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float pan { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float panLevel { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float pitch { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool playOnAwake { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int priority { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        [Obsolete("rolloffFactor is not supported anymore. Use min-, maxDist l rolloffMode instead.", true)]
        public float rolloffFactor { [MethodImpl(MethodImplOptions.InternalC WrapperlessIcall] get; [MethodImpl(MethodImplOptions. essI set; }

        public AudioRolloffMode rolloffMode {
```

```
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float spread { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

        public float time { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

        public int timeSamples { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }

        public AudioVelocityUpdateMode velocityUpdateMode {
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get;
[MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float volume { [MethodImpl(MethodImplOptions.InternalCall),
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
set; }
    }
}
```

回到顶部(go to top)

## UnityEngine.Light

```
namespace UnityEngine
{
    using System;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;

    public sealed class Light : Behaviour
    {
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public static extern Light[] GetLights(LightType type, int layer);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_areaSize(out Vector2 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_color(out Color value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_areaSize(ref Vector2 value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_color(ref Color value);

        public bool alreadyLightmapped {
[MethodImpl(MethodImplOptions.InternalCall), Wrapperl
[MethodImpl(MethodImplOptions.InternalCall), Wrapperl

```

推荐: 1

```
        public Vector2 areaSize
        {
            get
            {
                Vector2 vector;
                this.INTERNAL_get_areaSize(out vector);
                return vector;
            }
            set
            {
                this.INTERNAL_set_areaSize(ref value);
            }
        }

        [Obsolete("light.attenuate was removed; all lights always attenuate now", true)]
        public bool attenuate
        {
            get
            {
                return true;
            }
            set
            {
            }
        }

        public Color color
        {
            get
            {
                Color color;
                this.INTERNAL_get_color(out color);
                return color;
            }
            set
            {
                this.INTERNAL_set_color(ref value);
            }
        }

        public Texture cookie { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float cookieSize { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public int cullingMask { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public Flare flare { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.      essI
set; }

        public float intensity { [MethodImpl(Meth                     all),
```

推荐: 1

```
WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        [Obsolete("Use QualitySettings.pixelLightCount instead.")]
        public static int pixelLightCount { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float range { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public LightRenderMode renderMode { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float shadowBias { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        [Obsolete("light.shadowConstantBias was removed, use light.shadowBias", true)]
        public float shadowConstantBias
        {
            get
            {
                return 0f;
            }
            set
            {
            }
        }

        [Obsolete("light.shadowObjectSizeBias was removed, use light.shadowBias", true)]
        public float shadowObjectSizeBias
        {
            get
            {
                return 0f;
            }
            set
            {
            }
        }

        public LightShadows shadows { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float shadowSoftness { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessI
set; }

        public float shadowSoftnessFade { [MethodImpl(MethodImplOptions.InternalCall), Wrapperl
[MethodImpl(MethodImplOptions.InternalCall), Wrapperl

```

推荐: 1

```
        public float shadowStrength { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public float spotAngle { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public LightType type { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(go to top)

## UnityEngine.Animation

```
namespace UnityEngine
{
    using System;
    using System.Collections;
    using System.Reflection;
    using System.Runtime.CompilerServices;
    using System.Runtime.InteropServices;
    using UnityEngine.Internal;

    public sealed class Animation : Behaviour, IEnumerable
    {
        public void AddClip(AnimationClip clip, string newName)
        {
            this.AddClip(clip, newName, -2147483648, 0x7fffffff);
        }

        [ExcludeFromDocs]
        public void AddClip(AnimationClip clip, string newName, int firstFrame, int lastFrame)
        {
            bool addLoopFrame = false;
            this.AddClip(clip, newName, firstFrame, lastFrame, addLoopFrame);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void AddClip(AnimationClip clip, string newName, int firstFrame, int lastFrame, [DefaultValue("false")] bool addLoopFrame);
        [ExcludeFromDocs]
        public void Blend(string animation)
        {
            float fadeLength = 0.3f;
            float targetWeight = 1f;
            this.Blend(animation, targetWeight, f
        }
```

推荐: 1

```

        [ExcludeFromDocs]
        public void Blend(string animation, float targetWeight)
        {
            float fadeLength = 0.3f;
            this.Blend(animation, targetWeight, fadeLength);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Blend(string animation, [DefaultValue("1.0F")] float targetWeight, [DefaultValue("0.3F")] float fadeLength);
        [ExcludeFromDocs]
        public void CrossFade(string animation)
        {
            PlayMode stopSameLayer = PlayMode.StopSameLayer;
            float fadeLength = 0.3f;
            this.CrossFade(animation, fadeLength, stopSameLayer);
        }

        [ExcludeFromDocs]
        public void CrossFade(string animation, float fadeLength)
        {
            PlayMode stopSameLayer = PlayMode.StopSameLayer;
            this.CrossFade(animation, fadeLength, stopSameLayer);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void CrossFade(string animation, [DefaultValue("0.3F")] float fadeLength, [DefaultValue("PlayMode.StopSameLayer")] PlayMode mode);
        [ExcludeFromDocs]
        public AnimationState CrossFadeQueued(string animation)
        {
            PlayMode stopSameLayer = PlayMode.StopSameLayer;
            QueueMode completeOthers = QueueMode.CompleteOthers;
            float fadeLength = 0.3f;
            return this.CrossFadeQueued(animation, fadeLength, completeOthers, stopSameLayer);
        }

        [ExcludeFromDocs]
        public AnimationState CrossFadeQueued(string animation, float fadeLength)
        {
            PlayMode stopSameLayer = PlayMode.StopSameLayer;
            QueueMode completeOthers = QueueMode.CompleteOthers;
            return this.CrossFadeQueued(animation, fadeLength, completeOthers, stopSameLayer);
        }

        [ExcludeFromDocs]
        public AnimationState CrossFadeQueued(string animation, float fadeLength, QueueMode queue)
        {
            PlayMode stopSameLayer = PlayMode.StopSameLayer;
            return this.CrossFadeQueued(animation, fadeLength, queue, stopSameLayer);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern AnimationState CrossFadeQueued(string animation, [DefaultValue("0.3F")] float fadeLength, [DefaultValue("QueueMode.CompleteOthers")] QueueMode queue, [DefaultValue("PlayMode.StopSameLayer")] PlayMode mode);
        public AnimationClip GetClip(string name)
        {
            AnimationState state = this.GetState(name);
            if (state != null)
            {
                return state.clip;
            }
            return null;
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern int GetClipCount();
        public IEnumerator GetEnumerator()
        {
            return new Enumerator(this);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern AnimationState GetState(string name);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern AnimationState GetStateAtIndex(int index);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern int GetStateCount();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_Rewind(Animation self);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_Sample(Animation self);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_Stop(Animation self);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private static extern void INTERNAL_CALL_SyncLayer(Animation self, int layer);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_get_localBounds(out Bounds value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_RewindByName(string name);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void INTERNAL_set_localBounds(ref Bounds value);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_StopByName(string name);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern bool IsPlaying(string name);
        [ExcludeFromDocs]
        public bool Play()
        {
            PlayMode stopSameLayer = PlayMode.StopSameLayer;
            return this.Play(stopSameLayer);
        }

        [ExcludeFromDocs]
        public bool Play(string animation)
        {
            PlayMode stopSameLayer = PlayMode.Sto
            return this.Play(animation, stopSameL
        }

        [Obsolete("use PlayMode instead of AnimationPlayMode.")]
        public bool Play(AnimationPlayMode mode)
        {
            return this.PlayDefaultAnimation((PlayMode) mode);
        }

        public bool Play([DefaultValue("PlayMode.StopSameLayer")] PlayMode mode)
        {
            return this.PlayDefaultAnimation(mode);
        }

        [Obsolete("use PlayMode instead of AnimationPlayMode.")]
        public bool Play(string animation, AnimationPlayMode mode)
        {
            return this.Play(animation, (PlayMode) mode);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern bool Play(string animation,
[DefaultValue("PlayMode.StopSameLayer")] PlayMode mode);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool PlayDefaultAnimation(PlayMode mode);
        [ExcludeFromDocs]
        public AnimationState PlayQueued(string animation)
        {
            PlayMode stopSameLayer = PlayMode.StopSameLayer;
            QueueMode completeOthers = QueueMode.CompleteOthers;
            return this.PlayQueued(animation, completeOthers, stopSameLayer);
        }

        [ExcludeFromDocs]
        public AnimationState PlayQueued(string animation, QueueMode queue)
        {
            PlayMode stopSameLayer = PlayMode.StopSameLayer;
            return this.PlayQueued(animation, queue, stopSameLayer);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern AnimationState PlayQueued(string animation,
[DefaultValue("QueueMode.CompleteOthers")] QueueMode queue,
[DefaultValue("PlayMode.StopSameLayer")] PlayMode mode);
        public void RemoveClip(string clipName)
        {
            this.RemoveClip2(clipName);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void RemoveClip(AnimationClip clip);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void RemoveClip2(string clipName);
        public void Rewind()
        {
            INTERNAL_CALL_Rewind(this);
        }

        public void Rewind(string name)
        {
```

推荐: 1

```
            this.Internal_RewindByName(name);
        }

        public void Sample()
        {
            INTERNAL_CALL_Sample(this);
        }

        public void Stop()
        {
            INTERNAL_CALL_Stop(this);
        }

        public void Stop(string name)
        {
            this.Internal_StopByName(name);
        }

        public void SyncLayer(int layer)
        {
            INTERNAL_CALL_SyncLayer(this, layer);
        }

        [Obsolete("Use cullingType instead")]
        public bool animateOnlyIfVisible { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool animatePhysics { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public AnimationClip clip { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public AnimationCullingType cullingType { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public bool isPlaying { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; }

        public AnimationState this[string name]
        {
            get
            {
                return this.GetState(name);
            }
        }

        public Bounds localBounds
        {
            get
            {
                Bounds bounds;
                this.INTERNAL_get_localBounds(out
                return bounds;
            }
            set
            {
                this.INTERNAL_set_localBounds(ref value);
            }
        }

        public bool playAutomatically { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        public WrapMode wrapMode { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }

        private sealed class Enumerator : IEnumerator
        {
            private int m_CurrentIndex = -1;
            private Animation m_Outer;

            internal Enumerator(Animation outer)
            {
                this.m_Outer = outer;
            }

            public bool MoveNext()
            {
                int stateCount = this.m_Outer.GetStateCount();
                this.m_CurrentIndex++;
                return (this.m_CurrentIndex < stateCount);
            }

            public void Reset()
            {
                this.m_CurrentIndex = -1;
            }

            public object Current
            {
                get
                {
                    return this.m_Outer.GetStateAtIndex(this.m_CurrentIndex);
                }
            }
        }
    }
}
```

回到顶部(go to top)

## UnityEngine.MonoBehaviour

推荐: 1

```
namespace UnityEngine
{
    using System;
    using System.Collections;
    using System.Runtime.CompilerServices;
    using UnityEngine.Internal;

    public class MonoBehaviour : Behaviour
    {
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern MonoBehaviour();
        public void CancelInvoke()
        {
            this.Internal_CancelInvokeAll();
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void CancelInvoke(string methodName);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern void Internal_CancelInvokeAll();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        private extern bool Internal_IsInvokingAll();
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void Invoke(string methodName, float time);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void InvokeRepeating(string methodName, float time, float repeatRate);
        public bool IsInvoking()
        {
            return this.Internal_IsInvokingAll();
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern bool IsInvoking(string methodName);
        public static void print(object message)
        {
            Debug.Log(message);
        }

        public Coroutine StartCoroutine(IEnumerator routine)
        {
            return this.StartCoroutine_Auto(routine);
        }

        [ExcludeFromDocs]
        public Coroutine StartCoroutine(string methodName)
        {
            object obj2 = null;
            return this.StartCoroutine(methodName, obj2);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern Coroutine StartCoroutine(string methodName, [DefaultValue("null")] object value);
        [MethodImpl(MethodImplOptions.InternalCall
        public extern Coroutine StartCoroutine_Aut
        [MethodImpl(MethodImplOptions.InternalCall
        public extern void StopAllCoroutines();
        public void StopCoroutine(IEnumerator routine)
        {
            this.StopCoroutineViaEnumerator_Auto(routine);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        public extern void StopCoroutine(string methodName);
        public void StopCoroutine(Coroutine routine)
        {
            this.StopCoroutine_Auto(routine);
        }

        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern void StopCoroutine_Auto(Coroutine routine);
        [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall]
        internal extern void StopCoroutineViaEnumerator_Auto(IEnumerator routine);

        public bool useGUILayout { [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] get; [MethodImpl(MethodImplOptions.InternalCall), WrapperlessIcall] set; }
    }
}
```

回到顶部(go to top)

## 总结

标签: Unity3D

« 上一篇：自制Unity小游戏TankHero-2D(3)开始玩起来
» 下一篇：自制Unity小游戏TankHero-2D(4)关卡+小地图图标+碰撞条件分析

posted @ 2015-02-07 17:13 推荐: 1 (2) 编辑 收藏

**评论列表**

#1楼 2015-04-23 10:36 Good_good

这些源码怎么在Unity3D查看到呢？

支持(0) 反对(0)

#2楼[楼主] 2015-04-23 17:47 BIT祝威

@ Good_good

引用

这些源码怎么在Unity3D查看到呢？

reflector

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

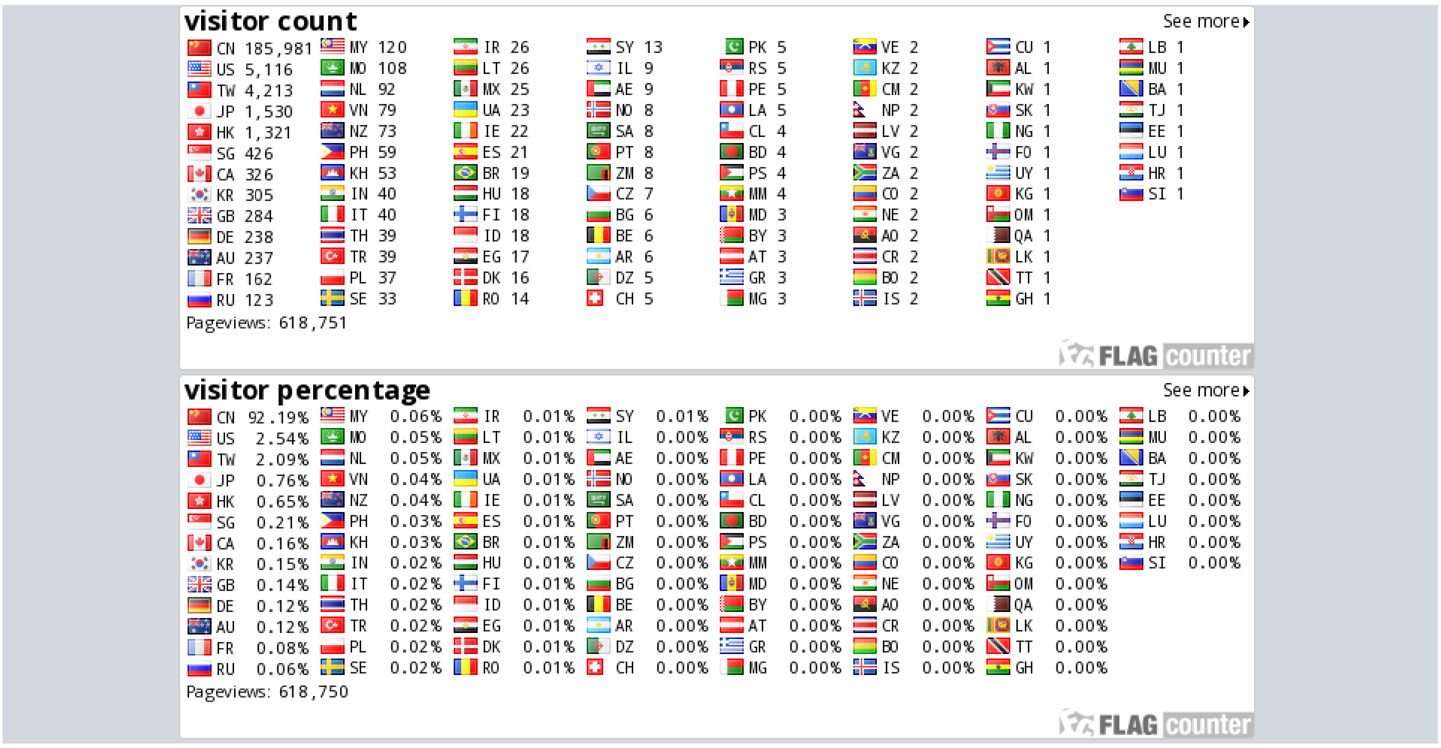
**注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。**

阿里云
海外节点全面降价 云产品6折起
BGP直连海外运营商骨干网 · 高速回流 · 免备案

**历史上的今天:**
2014-02-07 【OpenGL(SharpGL)】支持任意相机可平移缩放的轨迹球实现
2013-02-07 Winform游戏编程入门1:游戏循环的演化

Large Visitor Globe

推荐: 1



**visitor count** See more▸

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CN | 185,981 | MY | 120 | IR | 26 | SY | 13 | PK | 5 | VE | 2 | CU | 1 | LB | 1 |
| US | 5,116 | MO | 108 | LT | 26 | IL | 9 | RS | 5 | KZ | 2 | AL | 1 | MU | 1 |
| TW | 4,213 | NL | 92 | MX | 25 | AE | 9 | PE | 5 | CM | 2 | KW | 1 | BA | 1 |
| JP | 1,530 | VN | 79 | UA | 23 | NO | 8 | LA | 5 | NP | 2 | SK | 1 | TJ | 1 |
| HK | 1,321 | NZ | 73 | IE | 22 | SA | 8 | CL | 4 | LV | 2 | NG | 1 | EE | 1 |
| SG | 426 | PH | 59 | ES | 21 | PT | 8 | BD | 4 | VG | 2 | FO | 1 | LU | 1 |
| CA | 326 | KH | 53 | BR | 19 | ZM | 8 | PS | 4 | ZA | 2 | UY | 1 | HR | 1 |
| KR | 305 | IN | 40 | HU | 18 | CZ | 7 | MM | 4 | CO | 2 | KG | 1 | SI | 1 |
| GB | 284 | IT | 40 | FI | 18 | BG | 6 | MD | 3 | NE | 2 | OM | 1 | | |
| DE | 238 | TH | 39 | ID | 18 | BE | 6 | BY | 3 | AO | 2 | QA | 1 | | |
| AU | 237 | TR | 39 | EG | 17 | AR | 6 | AT | 3 | CR | 2 | LK | 1 | | |
| FR | 162 | PL | 37 | DK | 16 | DZ | 5 | GR | 3 | BO | 2 | TT | 1 | | |
| RU | 123 | SE | 33 | RO | 14 | CH | 5 | MG | 3 | IS | 2 | GH | 1 | | |

Pageviews: 618,751

FLAG counter

**visitor percentage** See more▸

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CN | 92.19% | MY | 0.06% | IR | 0.01% | SY | 0.01% | PK | 0.00% | VE | 0.00% | CU | 0.00% | LB | 0.00% |
| US | 2.54% | MO | 0.05% | LT | 0.01% | IL | 0.00% | RS | 0.00% | KZ | 0.00% | AL | 0.00% | MU | 0.00% |
| TW | 2.09% | NL | 0.05% | MX | 0.01% | AE | 0.00% | PE | 0.00% | CM | 0.00% | KW | 0.00% | BA | 0.00% |
| JP | 0.76% | VN | 0.04% | UA | 0.01% | NO | 0.00% | LA | 0.00% | NP | 0.00% | SK | 0.00% | TJ | 0.00% |
| HK | 0.65% | NZ | 0.04% | IE | 0.01% | SA | 0.00% | CL | 0.00% | LV | 0.00% | NG | 0.00% | EE | 0.00% |
| SG | 0.21% | PH | 0.03% | ES | 0.01% | PT | 0.00% | BD | 0.00% | VG | 0.00% | FO | 0.00% | LU | 0.00% |
| CA | 0.16% | KH | 0.03% | BR | 0.01% | ZM | 0.00% | PS | 0.00% | ZA | 0.00% | UY | 0.00% | HR | 0.00% |
| KR | 0.15% | IN | 0.02% | HU | 0.01% | CZ | 0.00% | MM | 0.00% | CO | 0.00% | KG | 0.00% | SI | 0.00% |
| GB | 0.14% | IT | 0.02% | FI | 0.01% | BG | 0.00% | MD | 0.00% | NE | 0.00% | OM | 0.00% | | |
| DE | 0.12% | TH | 0.02% | ID | 0.01% | BE | 0.00% | BY | 0.00% | AO | 0.00% | QA | 0.00% | | |
| AU | 0.12% | TR | 0.02% | EG | 0.01% | AR | 0.00% | AT | 0.00% | CR | 0.00% | LK | 0.00% | | |
| FR | 0.08% | PL | 0.02% | DK | 0.01% | DZ | 0.00% | GR | 0.00% | BO | 0.00% | TT | 0.00% | | |
| RU | 0.06% | SE | 0.02% | RO | 0.01% | CH | 0.00% | MG | 0.00% | IS | 0.00% | GH | 0.00% | | |

Pageviews: 618,750

FLAG counter

推荐: 1