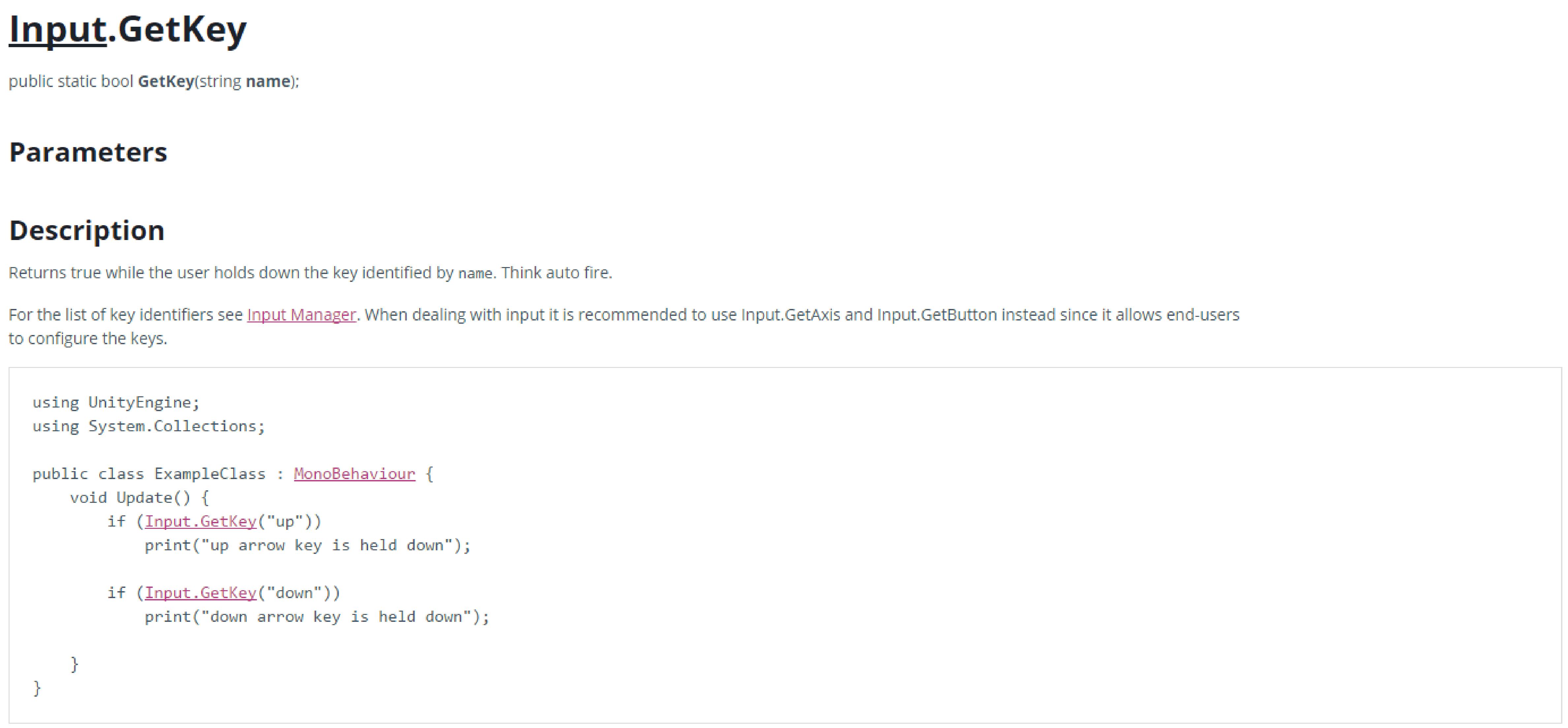
C#

(C Sharp)

Getting Input

(Quick Way)



Input.GetKey

public static bool **GetKey**(string **name**);

Parameters

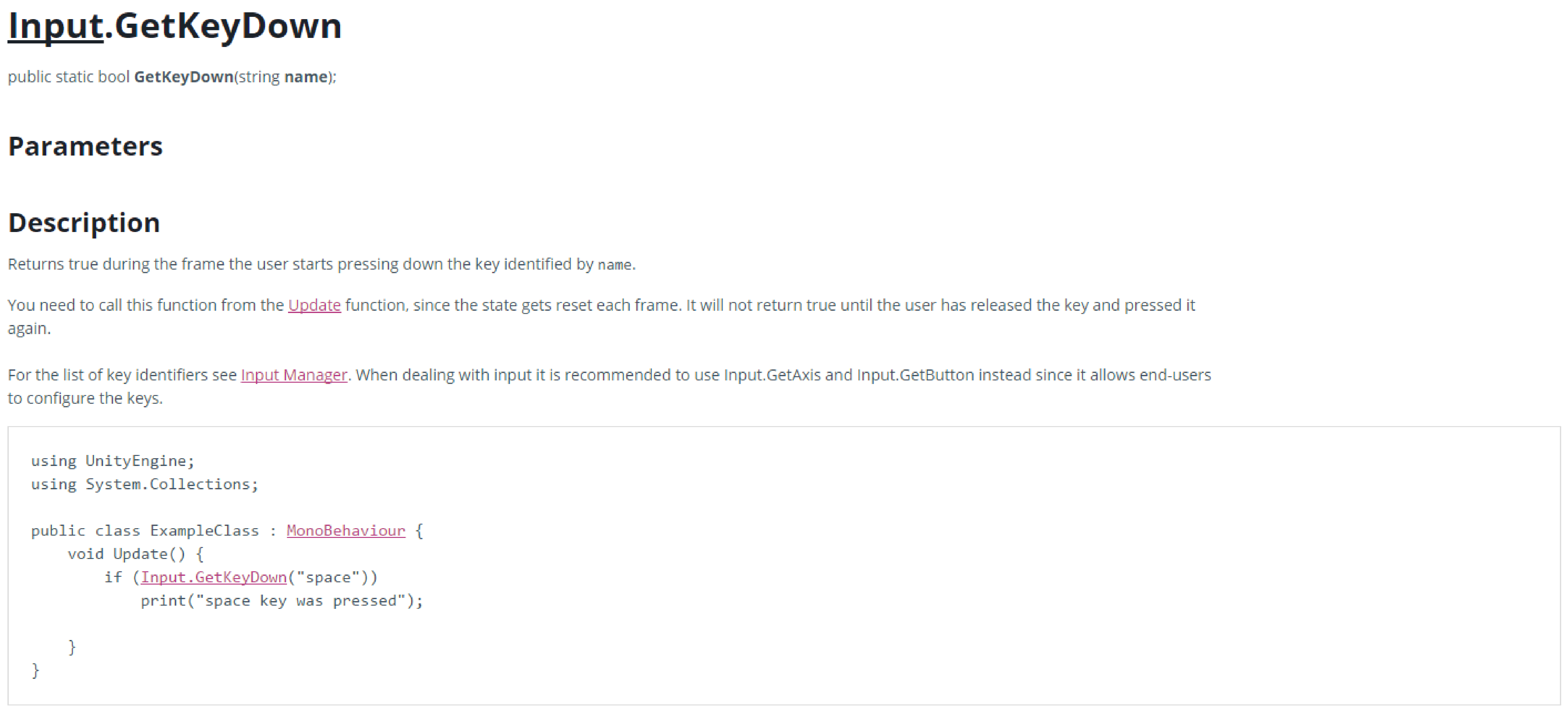
Description

Returns true while the user holds down the key identified by `name`. Think auto fire.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKey("up"))
            print("up arrow key is held down");

        if (Input.GetKey("down"))
            print("down arrow key is held down");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKey.html



Input.GetKeyDown

public static bool **GetKeyDown**(string **name**);

Parameters

Description

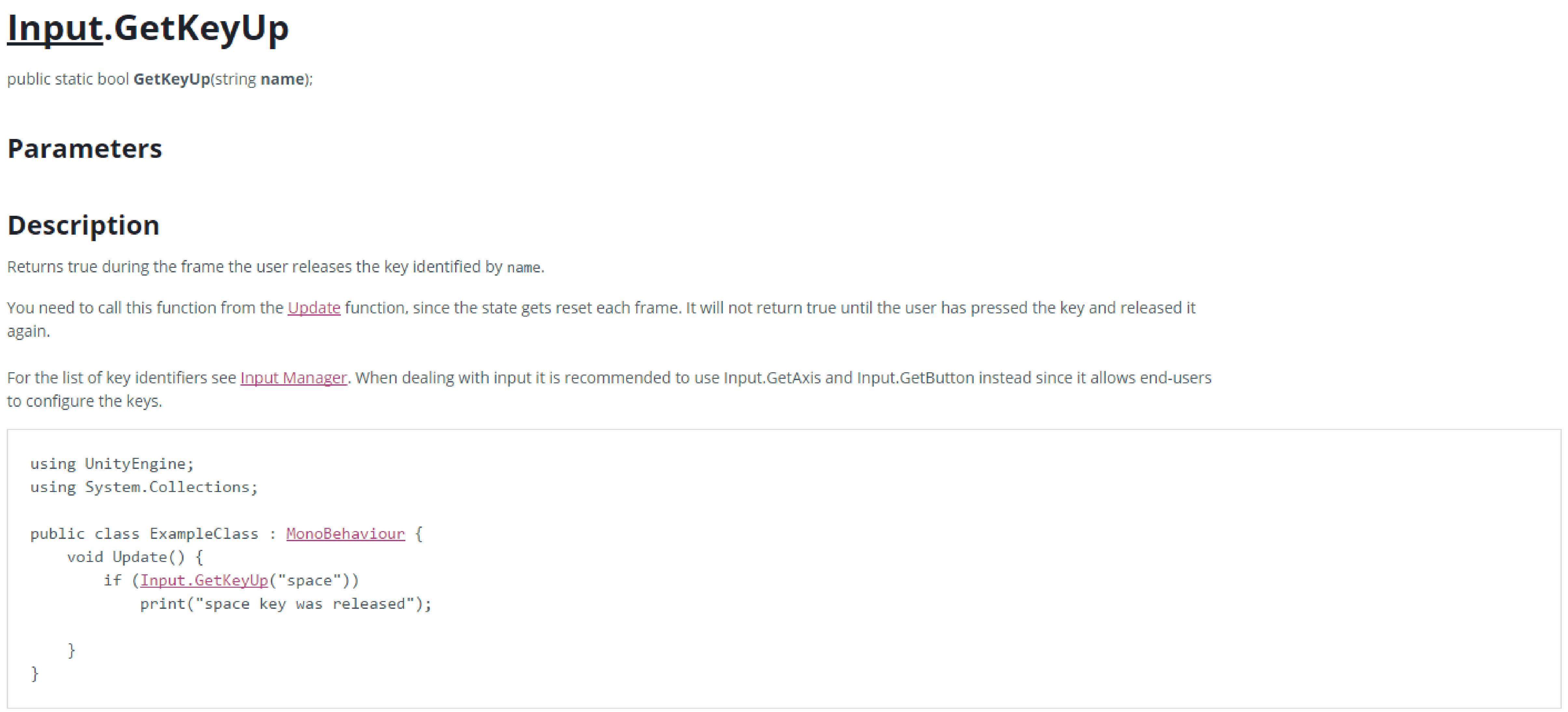
Returns true during the frame the user starts pressing down the key identified by `name`.

You need to call this function from the Update function, since the state gets reset each frame. It will not return true until the user has released the key and pressed it again.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKeyDown("space"))
            print("space key was pressed");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKeyDown.html



Input.GetKeyUp

public static bool **GetKeyUp**(string **name**);

Parameters

Description

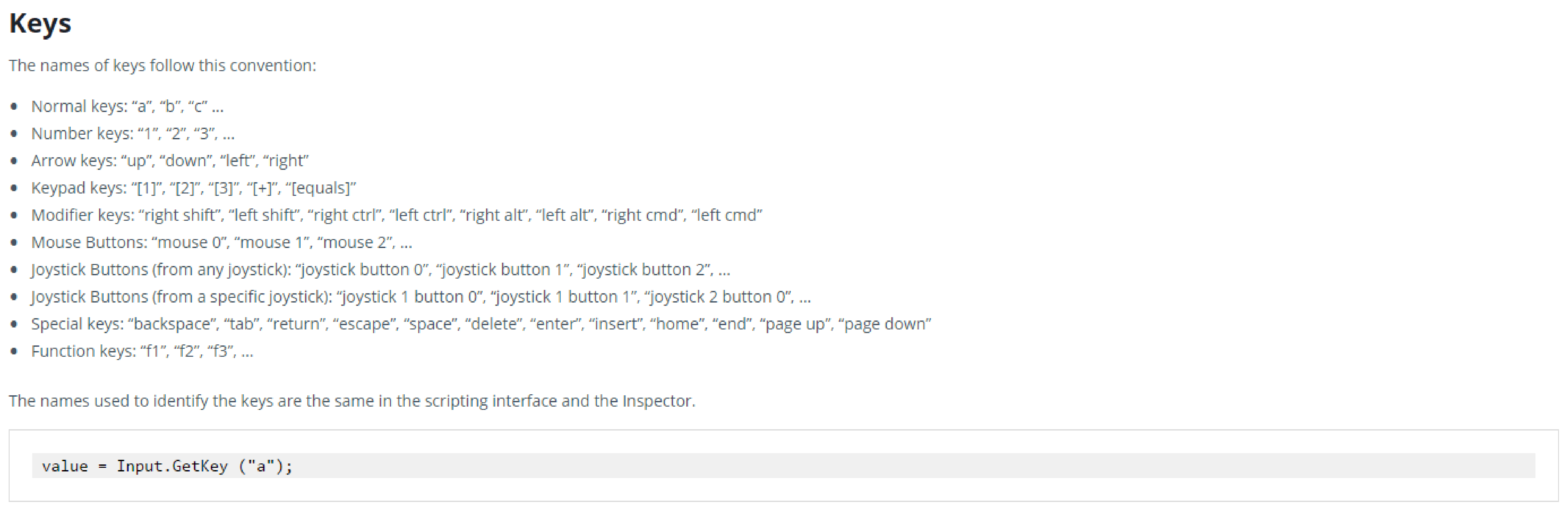
Returns true during the frame the user releases the key identified by `name`.

You need to call this function from the Update function, since the state gets reset each frame. It will not return true until the user has pressed the key and released it again.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKeyUp("space"))
            print("space key was released");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKeyUp.html



Keys

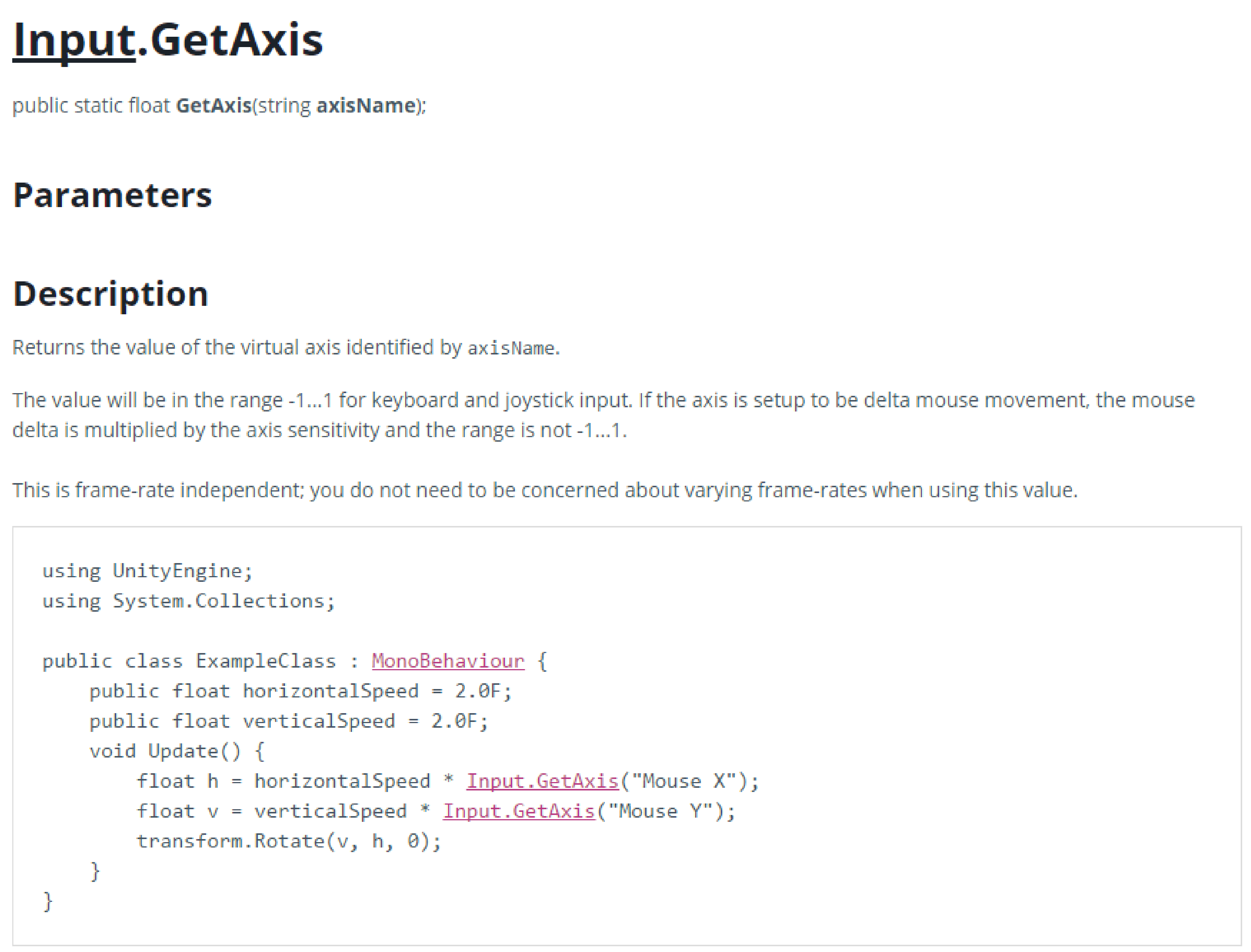
The names of keys follow this convention:

- Normal keys: “a”, “b”, “c” …
- Number keys: “1”, “2”, “3”, …
- Arrow keys: “up”, “down”, “left”, “right”
- Keypad keys: “[1]”, “[2]”, “[3]”, “[+]”, “[equals]”
- Modifier keys: “right shift”, “left shift”, “right ctrl”, “left ctrl”, “right alt”, “left alt”, “right cmd”, “left cmd”
- Mouse Buttons: “mouse 0”, “mouse 1”, “mouse 2”, …
- Joystick Buttons (from any joystick): “joystick button 0”, “joystick button 1”, “joystick button 2”, …
- Joystick Buttons (from a specific joystick): “joystick 1 button 0”, “joystick 1 button 1”, “joystick 2 button 0”, …
- Special keys: “backspace”, “tab”, “return”, “escape”, “space”, “delete”, “enter”, “insert”, “home”, “end”, “page up”, “page down”
- Function keys: “f1”, “f2”, “f3”, …

The names used to identify the keys are the same in the scripting interface and the Inspector.

```
value = Input.GetKey ("a");
```

https://docs.unity3d.com/Manual/ConventionalGameInput.html



Input.GetAxis

public static float **GetAxis**(string **axisName**);

Parameters

Description

Returns the value of the virtual axis identified by `axisName`.

The value will be in the range -1...1 for keyboard and joystick input. If the axis is setup to be delta mouse movement, the mouse delta is multiplied by the axis sensitivity and the range is not -1...1.

This is frame-rate independent; you do not need to be concerned about varying frame-rates when using this value.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public float horizontalSpeed = 2.0F;
    public float verticalSpeed = 2.0F;
    void Update() {
        float h = horizontalSpeed * Input.GetAxis("Mouse X");
        float v = verticalSpeed * Input.GetAxis("Mouse Y");
        transform.Rotate(v, h, 0);
    }
}
```

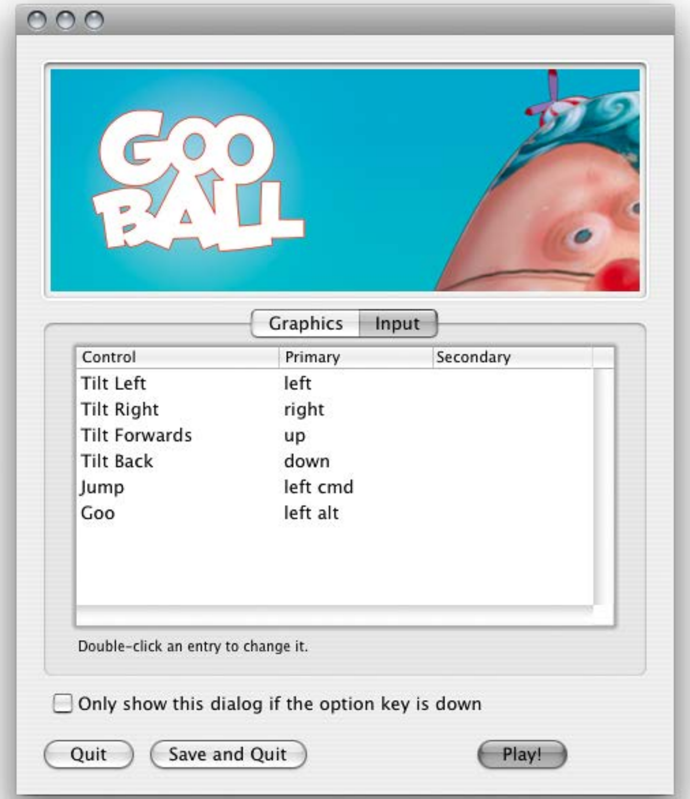
https://docs.unity3d.com/ScriptReference/Input.GetAxis.html

More Mouse Inputs

- Input.GetMouseButton
- Input.GetMouseButtonDown
- Input.GetMouseButtonUp

Customizable Input

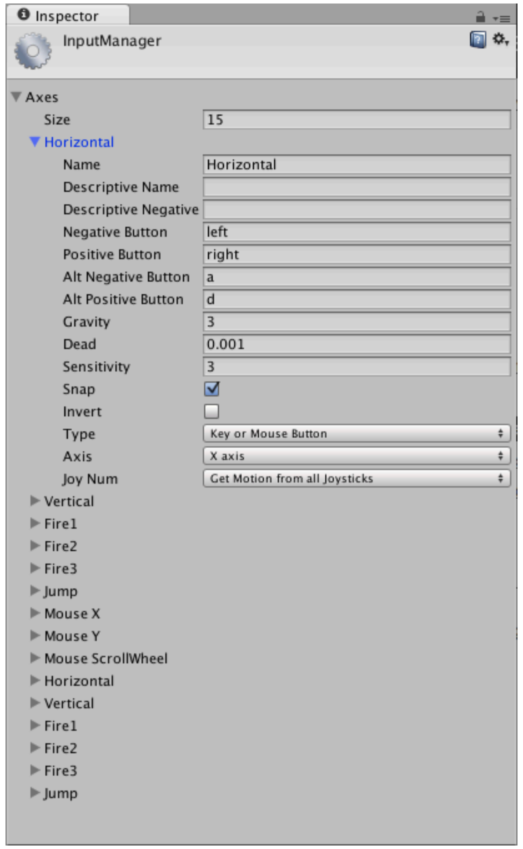
See https://docs.unity3d.com/Manual/Input.html



Control	Primary	Secondary
Tilt Left	left	
Tilt Right	right	
Tilt Forwards	up	
Tilt Back	down	
Jump	left cmd	
Goo	left alt	

Double-click an entry to change it.

☐ Only show this dialog if the option key is down



InputManager

Axes

Size	15

Horizontal

Name	Horizontal
Descriptive Name	
Descriptive Negative	
Negative Button	left
Positive Button	right
Alt Negative Button	a
Alt Positive Button	d
Gravity	3
Dead	0.001
Sensitivity	3
Snap	✓
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

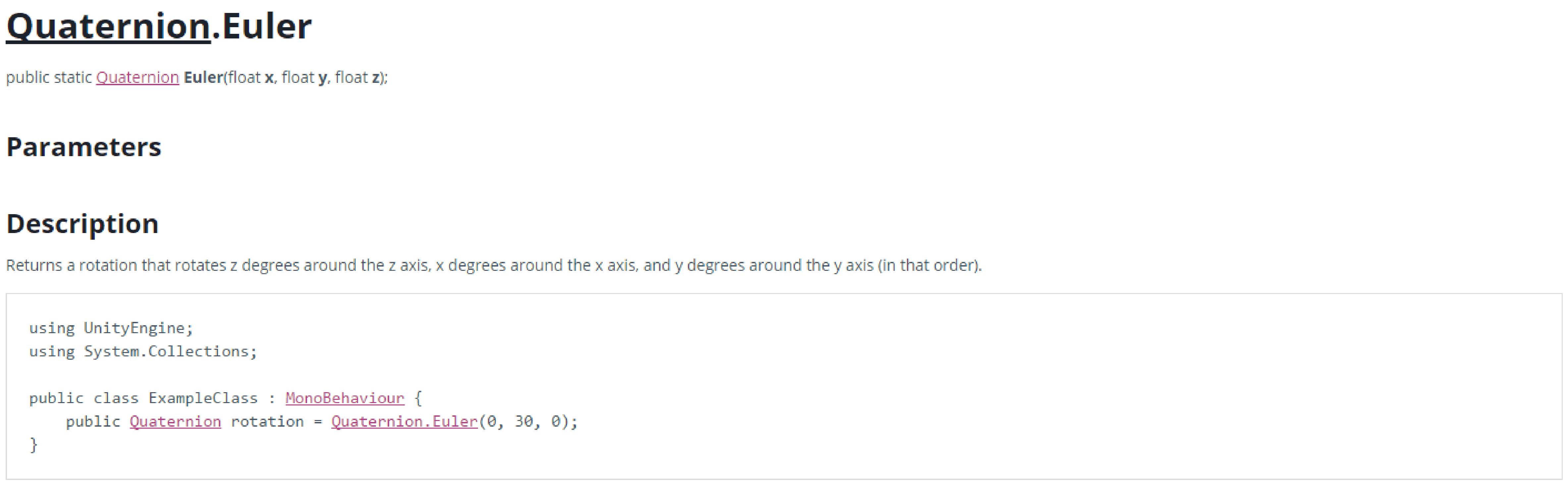
- Vertical
- Fire1
- Fire2
- Fire3
- Jump
- Mouse X
- Mouse Y
- Mouse ScrollWheel
- Horizontal
- Vertical
- Fire1
- Fire2
- Fire3
- Jump

Euler vs Quaternions

Euler Rotation

```
float horizontalMovement = Input.GetAxis("Mouse X");
float verticalMovement = Input.GetAxis("Mouse Y");

// Wrong way to rotate along two axes! Don't do this.
transform.Rotate(0, horizontalMovement, 0);
transform.Rotate(-verticalMovement, 0, 0);
```

Gimbal Lock
in 30 seconds



Quaternion.Euler

public static Quaternion **Euler**(float **x**, float **y**, float **z**);

Parameters

Description

Returns a rotation that rotates z degrees around the z axis, x degrees around the x axis, and y degrees around the y axis (in that order).

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Quaternion rotation = Quaternion.Euler(0, 30, 0);
}
```

https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html

Quaternion Rotation

```
// Rotating with quaternions - much better!
transform.localRotation = Quaternion.Euler(45f, 20f, 0f);
```

```
// Exercise:
//
//  - Add forward/backward movement with the w and s keys
//  - Add strafing movement (left/right) with the a and d keys
//  - Add vertical movement (up/down) with the q and e keys
//  - Bonus: add the ability to hold shift to move at 2x speed
//
// Note: Use a public field for Speed, so these movement speeds can be
// adjusted in the inspector. Speed should be in meters/second.
```

Classes and Instances

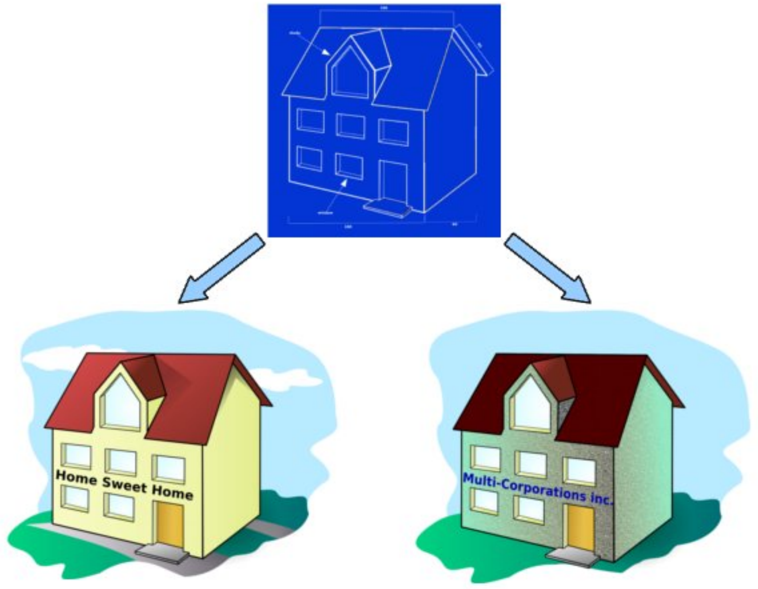
(OOP)

Classes & Instances

- Encapsulation: organize variables and functions together
- Code reuse

Analogy Time

- Blueprint -> House
- Cookie Cutter -> Cookie
- Person -> Bob



http://processing.lyndondaniels.com/53blueprint.php

Class

ACCESS MODIFIER

CLASS NAME

```
public class Enemy {
    // Fields and methods go inside brackets
}
```

Fields

Fields

```
public class Enemy {
    // Fields
    public string Name;
}
```

$\underbrace{\texttt{public}}_{\text{ACCESS MODIFIER}}\ \underbrace{\texttt{string}}_{\text{VARIABLE TYPE}}\ \underbrace{\texttt{Name}}_{\text{VARIABLE NAME}}$

```
public class ClassDemo : MonoBehaviour {

    void Start () {
        Enemy enemy1 = new Enemy();
    }

}

public class Enemy {
    // Fields
    public string Name;
}
```

INSTANCE

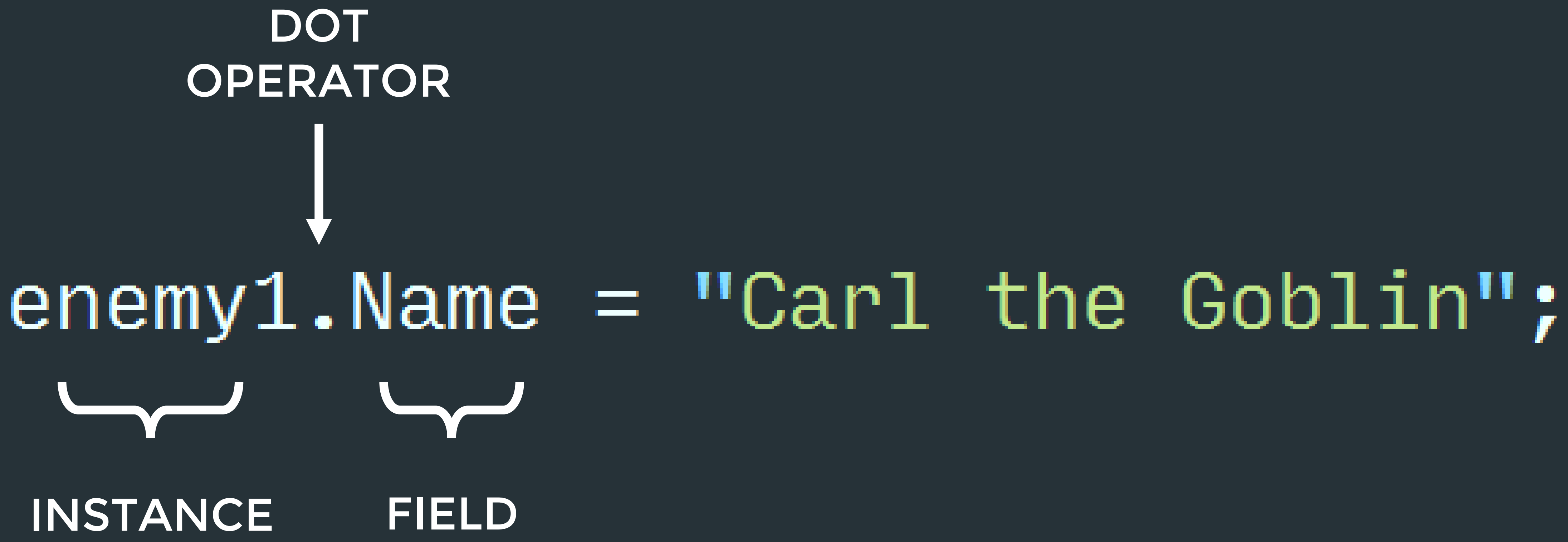
CLASS

$$\underbrace{\texttt{Enemy}}_{\text{VARIABLE TYPE}}\ \overbrace{\texttt{enemy1}}^{\text{VARIABLE NAME}}\ \texttt{=}\ \underbrace{\textit{\texttt{new}}}_{\text{KEYWORD}}\ \overbrace{\texttt{Enemy()}}^{\text{CONSTRUCTOR}}\texttt{;}$$

INSTANCE →

CLASS {

```
public class ClassDemo : MonoBehaviour {

    void Start () {
        Enemy enemy1 = new Enemy();
        enemy1.Name = "Carl The Goblin";
        Debug.Log("Monster 1 is: " + enemy1.Name);
    }

}

public class Enemy {
    // Fields
    public string Name;
}
```



```
enemy1.Name = "Carl the Goblin";
```

```
public class ClassDemo : MonoBehaviour {

    void Start () {
```

INSTANCE →

```
        Enemy enemy1 = new Enemy();
        enemy1.Name = "Carl The Goblin";
        Debug.Log("Monster 1 is: " + enemy1.Name);
```

INSTANCE →

```
        Enemy enemy2 = new Enemy();
        enemy2.Name = "Radcliff";
        Debug.Log("Monster 2 is: " + enemy2.Name);
    }

}
```

CLASS

```
public class Enemy {
    // Fields
    public string Name;
}
```

Constructors

```
public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }
}
```

```
public class ClassDemo : MonoBehaviour {
    void Start () {
        Enemy enemy1 = new Enemy("Carl The Goblin");
        Debug.Log("Enemy 1 is: " + enemy1.Name);
    }
}

public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }
}
```

Methods

```csharp
public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }

    // Methods
    public void Speak() {
        Debug.Log("Hello, I am " + Name + ".");
    }
}
```

```
Enemy enemy1 = new Enemy("Carl The Goblin");
enemy1.Speak();
```

More Methods

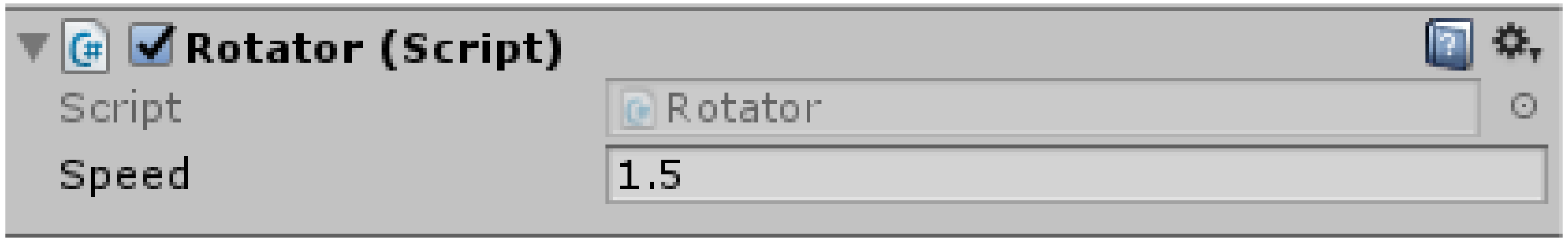
```
public class Enemy {
    // Fields
    public string Name;
    private int Health;

    // Constructor
    public Enemy(string name, int health) {
        Name = name;
        Health = health;
    }

    // Methods
    public void Speak() {
        Debug.Log("Hello, I am " + Name + ".");
    }
    public void TakeDamage(int damage) {
        Health = Health - damage;
    }
    public void SayHealth() {
        if (Health < 0) {
            Debug.Log(Name + ": I am dead :(");
        } else {
            Debug.Log(Name + ": I have " + Health + " health");
        }
    }
}
```

```
// Create a Rectangle class:
//  - It should have public fields for Width & Height
//  - It should have a constructor that can initialize the Width and Height
//  - It should have a method GetArea that returns a float
//  - It should have a method GetPerimeter that returns a float
//
// Test it by creating a couple Rectangle instances and calculating the area
// and perimeter
```

Scripts are Classes

```
public class Rotator : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

```
public class Rotator : MonoBehaviour {

    public float speed;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        float rotationAmount = speed * Time.deltaTime;
        transform.Rotate(rotationAmount, 0f, 0f);
    }
}
```



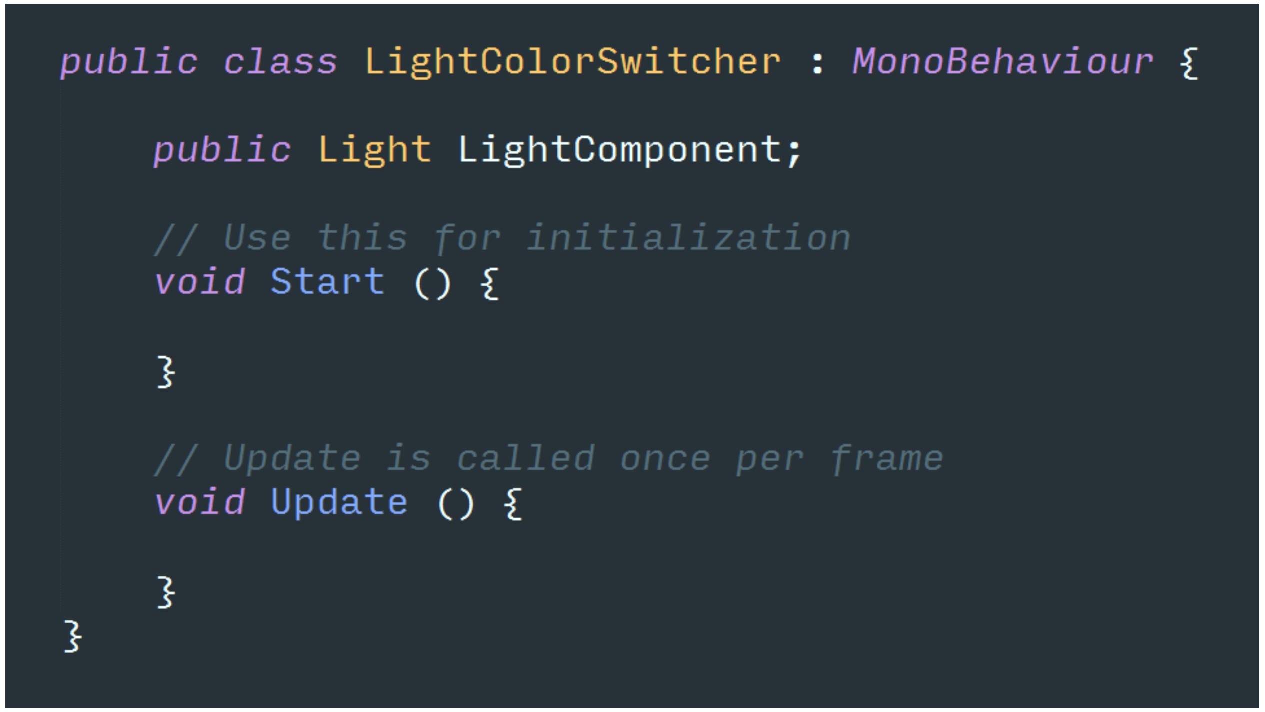
Script Rotator

Speed 1.5

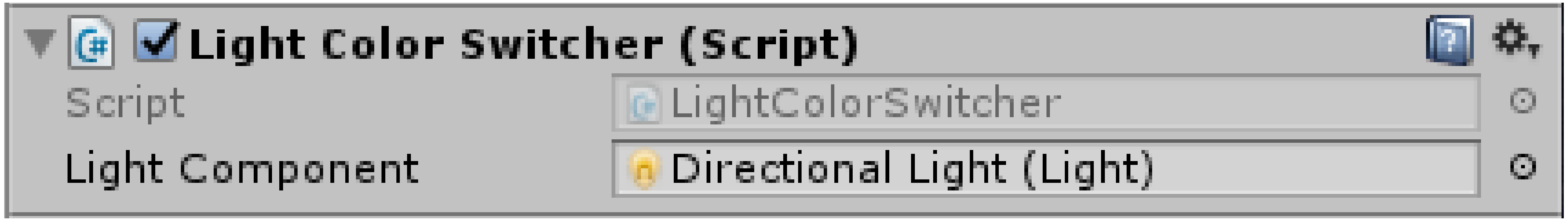
```
public class Rotator : MonoBehaviour {

    public float speed = 10;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        float rotationAmount = speed * Time.deltaTime;
        transform.Rotate(rotationAmount, 0f, 0f);
    }
}
```

Accessing Components

Via Inspector

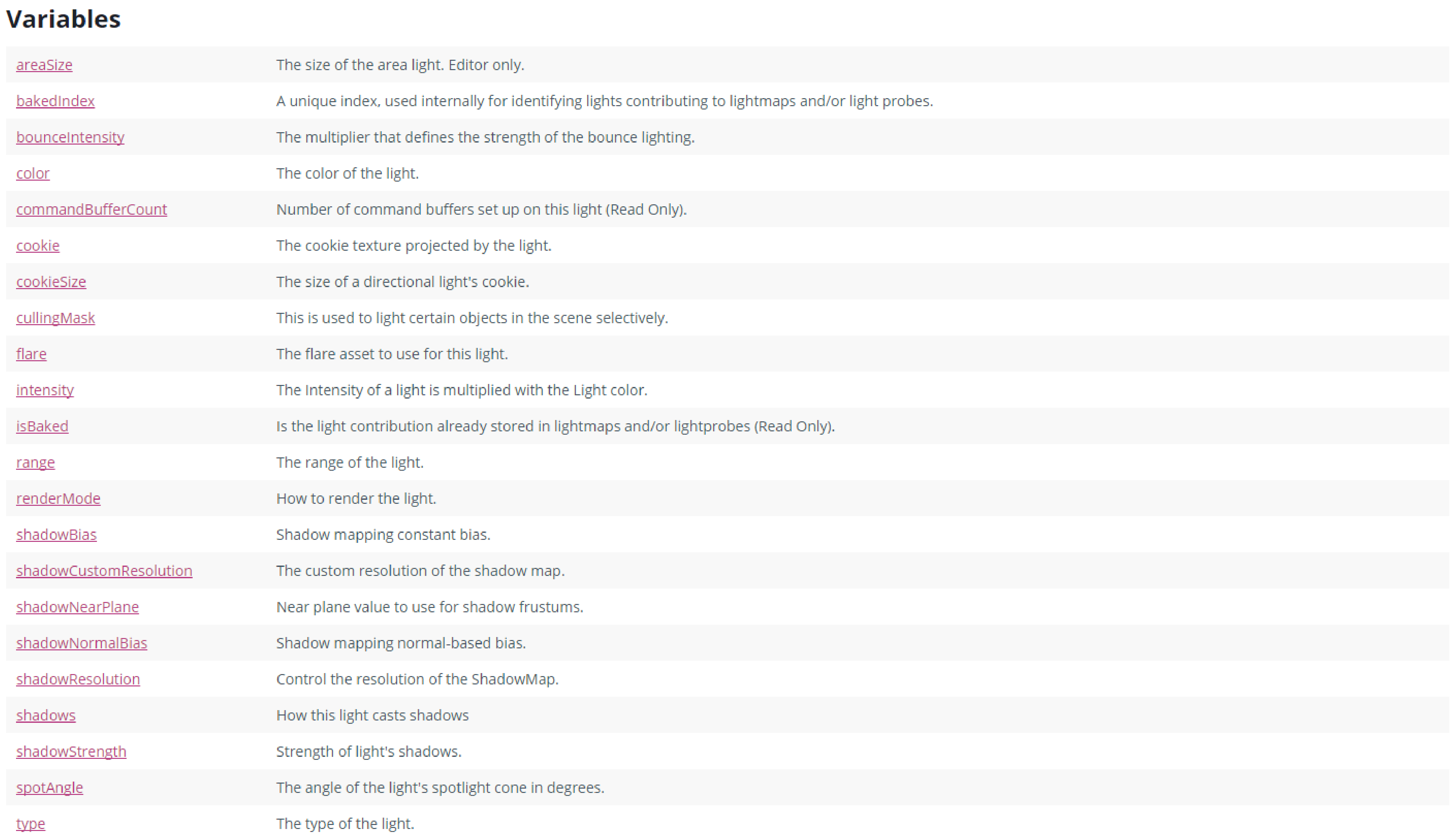


```
public class LightColorSwitcher : MonoBehaviour {

    public Light LightComponent;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```



Light Color Switcher (Script)

Script	LightColorSwitcher
Light Component	Directional Light (Light)

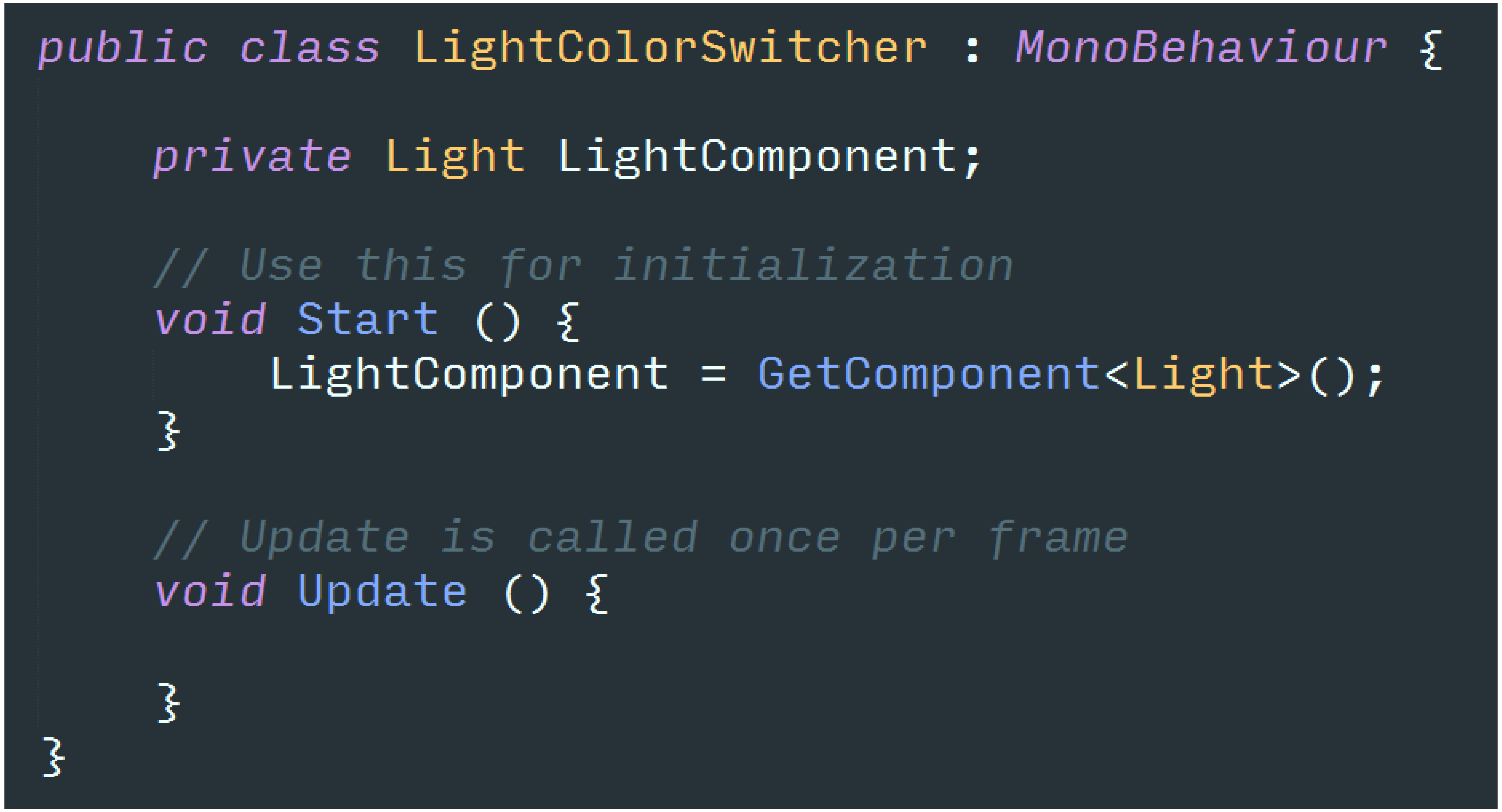


Variables

areaSize	The size of the area light. Editor only.
bakedIndex	A unique index, used internally for identifying lights contributing to lightmaps and/or light probes.
bounceIntensity	The multiplier that defines the strength of the bounce lighting.
color	The color of the light.
commandBufferCount	Number of command buffers set up on this light (Read Only).
cookie	The cookie texture projected by the light.
cookieSize	The size of a directional light's cookie.
cullingMask	This is used to light certain objects in the scene selectively.
flare	The flare asset to use for this light.
intensity	The Intensity of a light is multiplied with the Light color.
isBaked	Is the light contribution already stored in lightmaps and/or lightprobes (Read Only).
range	The range of the light.
renderMode	How to render the light.
shadowBias	Shadow mapping constant bias.
shadowCustomResolution	The custom resolution of the shadow map.
shadowNearPlane	Near plane value to use for shadow frustums.
shadowNormalBias	Shadow mapping normal-based bias.
shadowResolution	Control the resolution of the ShadowMap.
shadows	How this light casts shadows
shadowStrength	Strength of light's shadows.
spotAngle	The angle of the light's spotlight cone in degrees.
type	The type of the light.

http://docs.unity3d.com/ScriptReference/Light.html

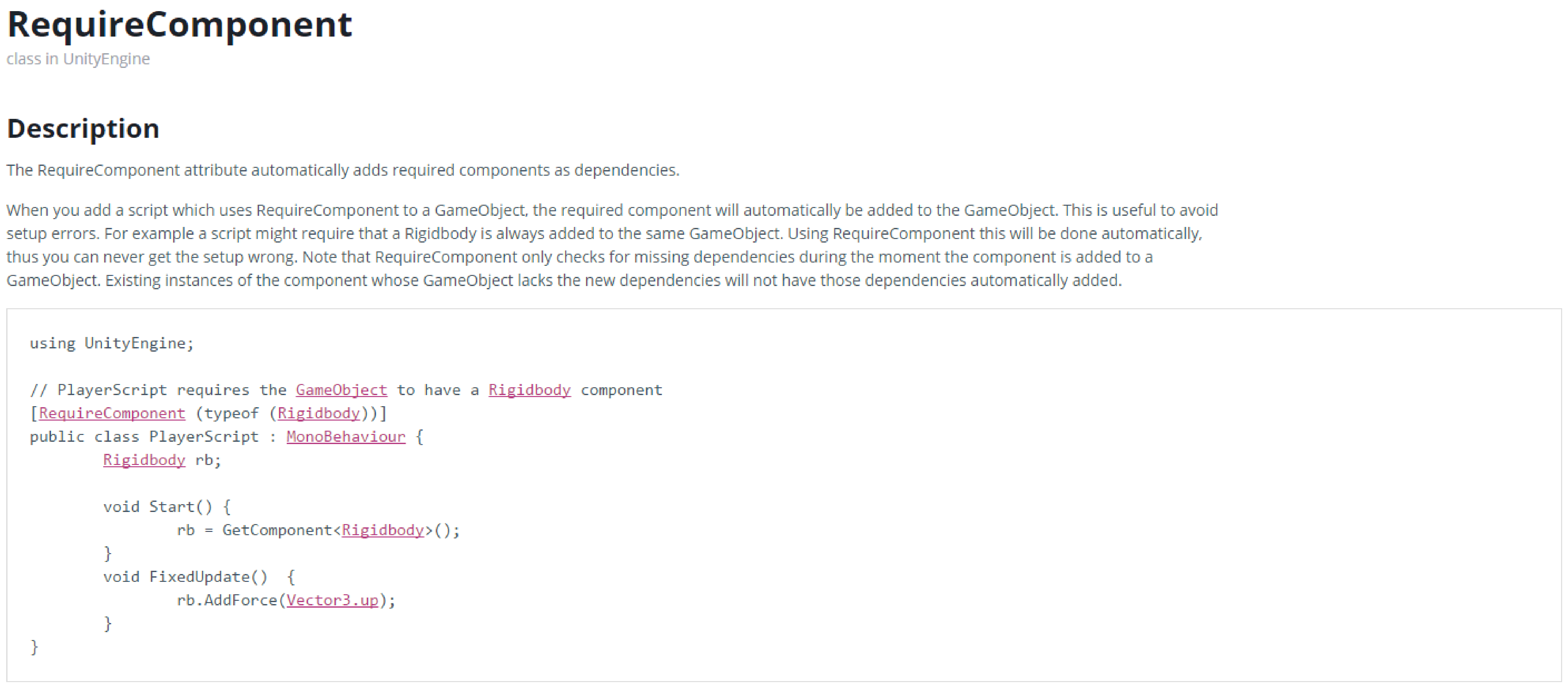
Via Scripting


```
public class LightColorSwitcher : MonoBehaviour {

    private Light LightComponent;

    // Use this for initialization
    void Start () {
        LightComponent = GetComponent<Light>();
    }

    // Update is called once per frame
    void Update () {

    }
}
```

Generic Method

```
LightComponent = GetComponent<Light>();
```

TYPE OF COMPONENT



RequireComponent

class in UnityEngine

Description

The RequireComponent attribute automatically adds required components as dependencies.

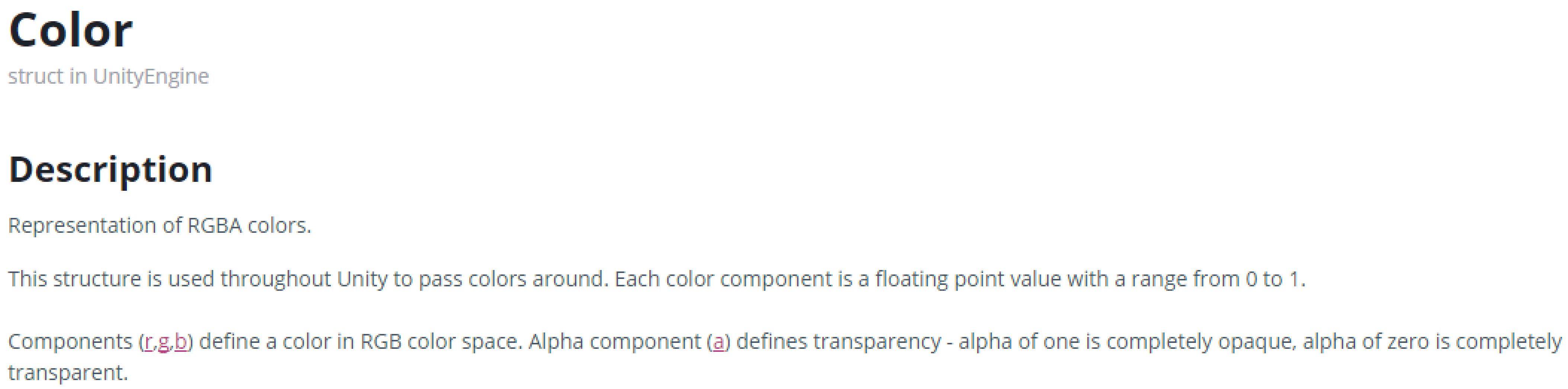
When you add a script which uses RequireComponent to a GameObject, the required component will automatically be added to the GameObject. This is useful to avoid setup errors. For example a script might require that a Rigidbody is always added to the same GameObject. Using RequireComponent this will be done automatically, thus you can never get the setup wrong. Note that RequireComponent only checks for missing dependencies during the moment the component is added to a GameObject. Existing instances of the component whose GameObject lacks the new dependencies will not have those dependencies automatically added.

```
using UnityEngine;

// PlayerScript requires the GameObject to have a Rigidbody component
[RequireComponent (typeof (Rigidbody))]
public class PlayerScript : MonoBehaviour {
        Rigidbody rb;

        void Start() {
                rb = GetComponent<Rigidbody>();
        }
        void FixedUpdate()  {
                rb.AddForce(Vector3.up);
        }
}
```

https://docs.unity3d.com/ScriptReference/RequireComponent.html

Color

http://docs.unity3d.com/ScriptReference/Color.html



Color

struct in UnityEngine

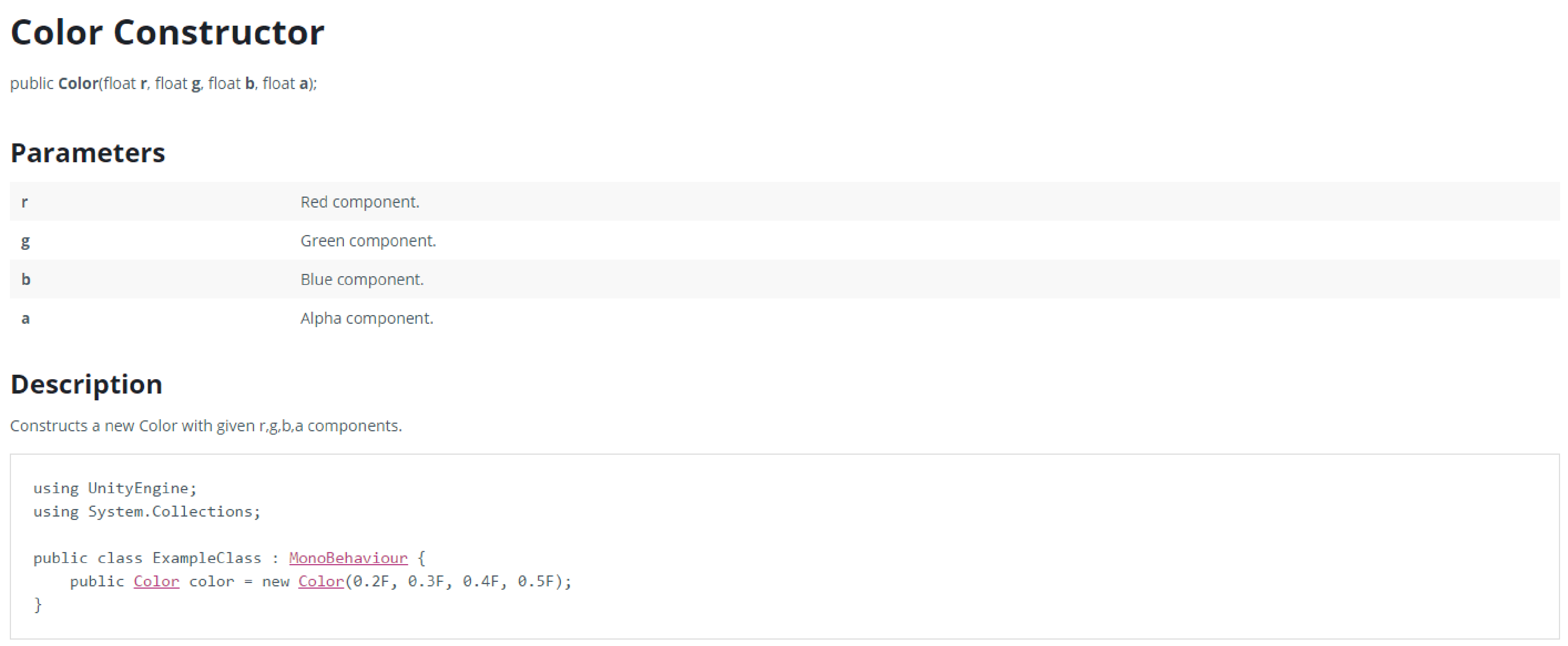
Description

Representation of RGBA colors.

This structure is used throughout Unity to pass colors around. Each color component is a floating point value with a range from 0 to 1.

Components (r,g,b) define a color in RGB color space. Alpha component (a) defines transparency - alpha of one is completely opaque, alpha of zero is completely transparent.

https://docs.unity3d.com/ScriptReference/Color.html



Color Constructor

public **Color**(float **r**, float **g**, float **b**, float **a**);

Parameters

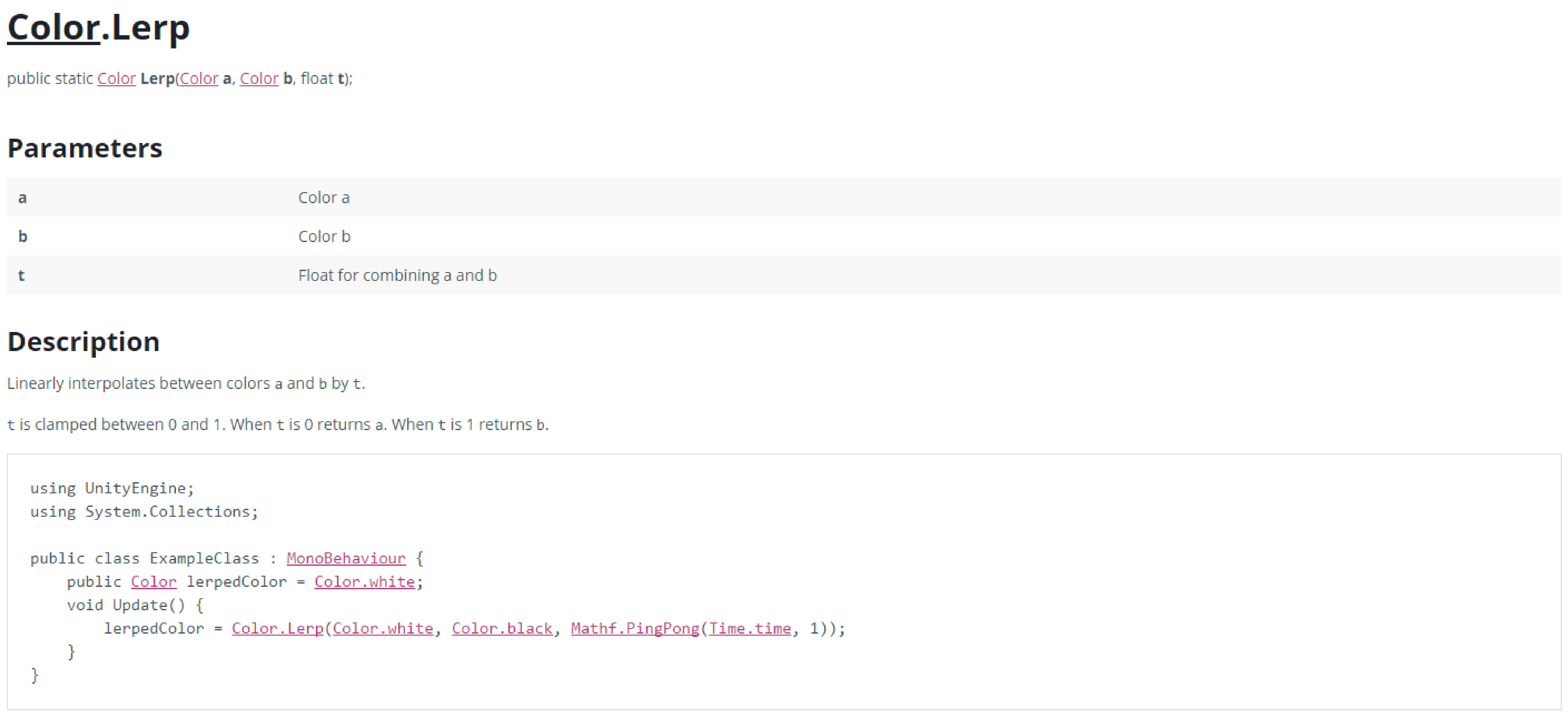
r	Red component.
g	Green component.
b	Blue component.
a	Alpha component.

Description

Constructs a new Color with given r,g,b,a components.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Color color = new Color(0.2F, 0.3F, 0.4F, 0.5F);
}
```

https://docs.unity3d.com/ScriptReference/Color-ctor.html



Color.Lerp

public static Color **Lerp**(Color **a**, Color **b**, float **t**);

Parameters

a	Color a
b	Color b
t	Float for combining a and b

Description

Linearly interpolates between colors a and b by t.

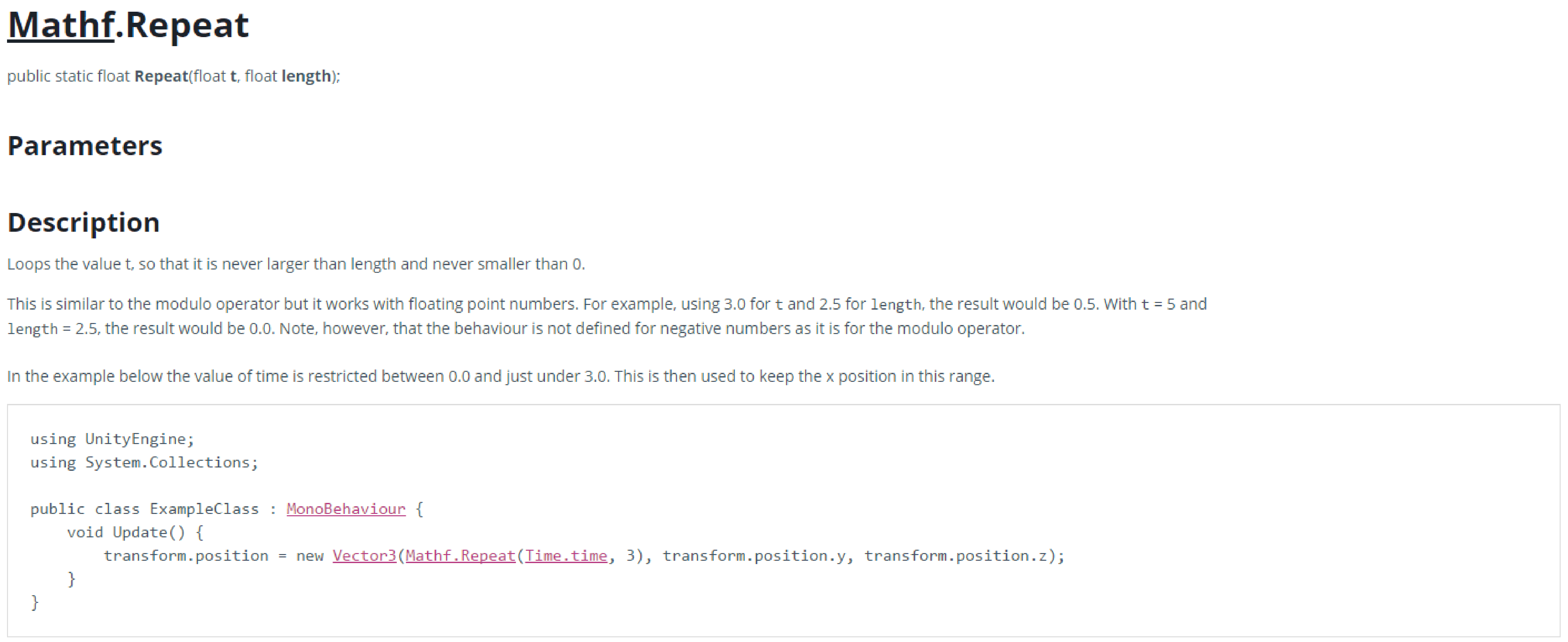
t is clamped between 0 and 1. When t is 0 returns a. When t is 1 returns b.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Color lerpedColor = Color.white;
    void Update() {
        lerpedColor = Color.Lerp(Color.white, Color.black, Mathf.PingPong(Time.time, 1));
    }
}
```

http://docs.unity3d.com/ScriptReference/Color.Lerp.html

Mathf

http://docs.unity3d.com/ScriptReference/Mathf.html



Mathf.Repeat

public static float **Repeat**(float **t**, float **length**);

Parameters

Description

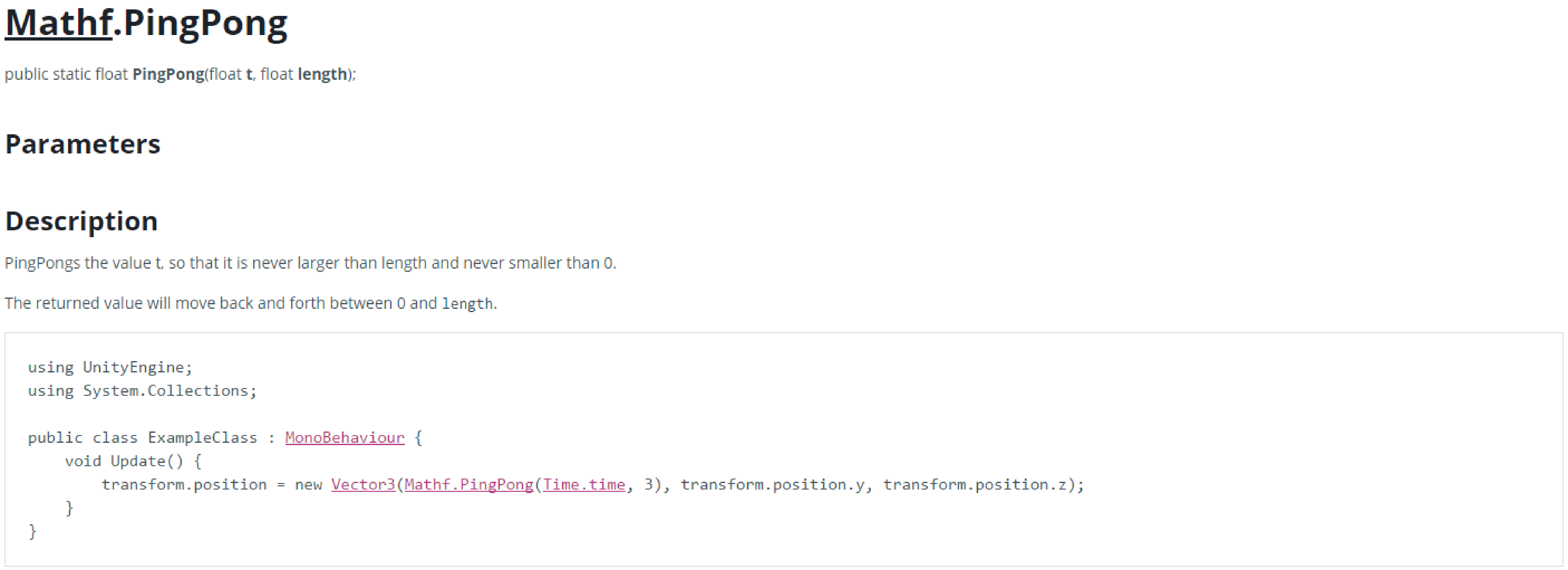
Loops the value t, so that it is never larger than length and never smaller than 0.

This is similar to the modulo operator but it works with floating point numbers. For example, using 3.0 for `t` and 2.5 for `length`, the result would be 0.5. With `t` = 5 and `length` = 2.5, the result would be 0.0. Note, however, that the behaviour is not defined for negative numbers as it is for the modulo operator.

In the example below the value of time is restricted between 0.0 and just under 3.0. This is then used to keep the x position in this range.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.position = new Vector3(Mathf.Repeat(Time.time, 3), transform.position.y, transform.position.z);
    }
}
```

http://docs.unity3d.com/ScriptReference/Mathf.Repeat.html



Mathf.PingPong

public static float **PingPong**(float **t**, float **length**);

Parameters

Description

PingPongs the value t, so that it is never larger than length and never smaller than 0.

The returned value will move back and forth between 0 and `length`.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.position = new Vector3(Mathf.PingPong(Time.time, 3), transform.position.y, transform.position.z);
    }
}
```

http://docs.unity3d.com/ScriptReference/Mathf.PingPong.html

Vector3

http://docs.unity3d.com/ScriptReference/Vector3.html

Vector3

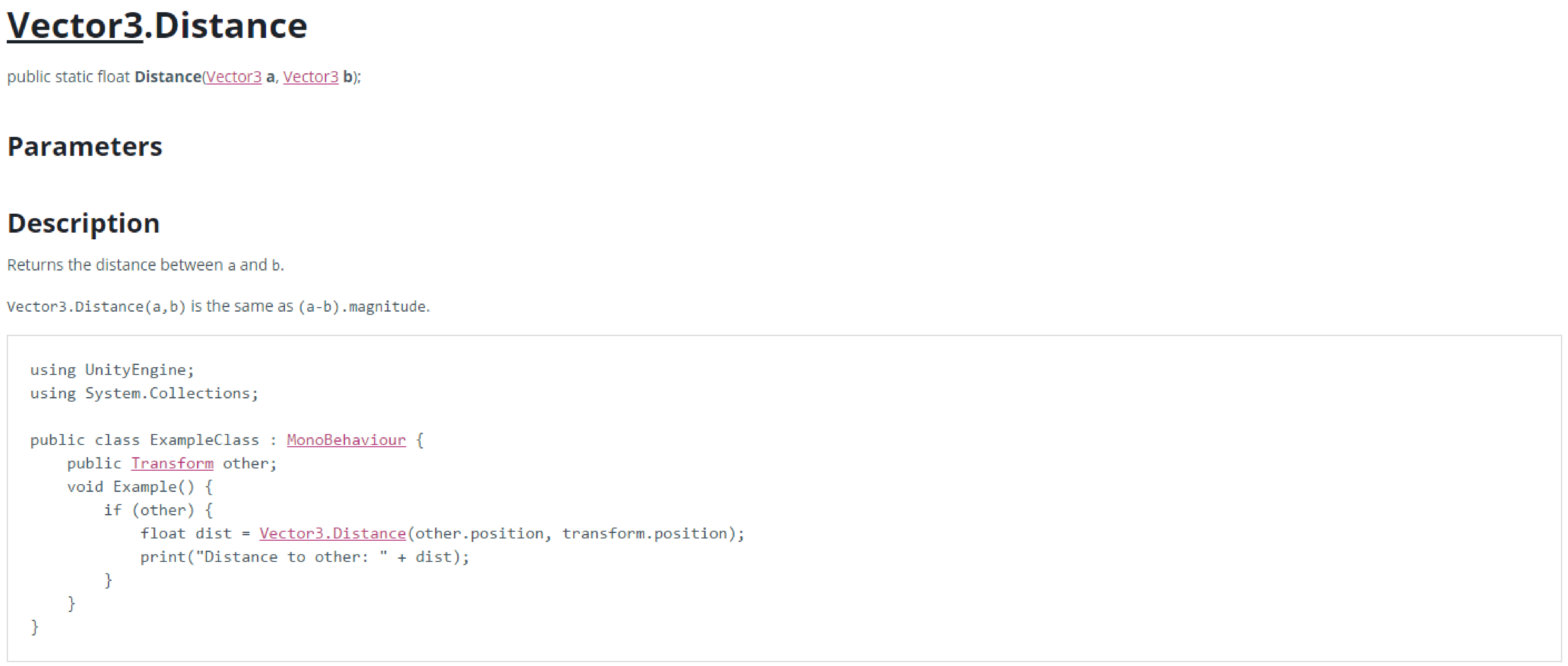
struct in UnityEngine

Description

Representation of 3D vectors and points.

This structure is used throughout Unity to pass 3D positions and directions around. It also contains functions for doing common vector operations.

```
Vector3 position = new Vector3(0f, 0f, 1f);
```



Vector3.Distance

public static float **Distance**(Vector3 **a**, Vector3 **b**);

Parameters

Description

Returns the distance between `a` and `b`.

`Vector3.Distance(a,b)` is the same as `(a-b).magnitude`.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Transform other;
    void Example() {
        if (other) {
            float dist = Vector3.Distance(other.position, transform.position);
            print("Distance to other: " + dist);
        }
    }
}
```

http://docs.unity3d.com/ScriptReference/Vector3.Distance.html

```
public class DistanceDemo : MonoBehaviour {

    public Transform PlayerTransform;

    void Update () {

        // Find the distance
        float distance = Vector3.Distance(PlayerTransform.position, transform.position);

        // Check how this object is to the player
        if (distance <= 3f) {
            Debug.Log("Player is close!");
        } else {
            Debug.Log("Player is far!");
        }
    }
}
```

MeshRenderer & Material

http://docs.unity3d.com/ScriptReference/MeshRenderer.html
http://docs.unity3d.com/ScriptReference/Material.html

```
public class GettingMaterial : MonoBehaviour {

    private MeshRenderer renderer;
    private Material mat;

    void Start () {
        // Get the MeshRenderer on this game object
        renderer = GetComponent<MeshRenderer>();
        // Get the first material from the renderer
        mat = renderer.material;
        // Change the material's color to red
        mat.color = new Color(1f, 0f, 0f);
    }
}
```