

# INDEX

## VectorLine

|  |   |
|--|---|
| <a href="#">VectorLine (constructor)</a> ..... | 2 |
| <a href="#">active</a> .....                   | 3 |
| <a href="#">capLength</a> .....                | 3 |
| <a href="#">continuous</a> .....               | 3 |
| <a href="#">depth</a> .....                    | 3 |
| <a href="#">layer</a> .....                    | 4 |
| <a href="#">lineWidth</a> .....                | 4 |
| <a href="#">maxDrawIndex</a> .....             | 4 |
| <a href="#">minDrawIndex</a> .....             | 4 |
| <a href="#">name</a> .....                     | 4 |
| <a href="#">smoothWidth</a> .....              | 5 |
| <a href="#">Resize</a> .....                   | 5 |

## VectorPoints

|  |   |
|--|---|
| <a href="#">VectorPoints (constructor)</a> ..... | 6 |
|--|---|

## Vector functions and variables

|   |    |
|---|----|
| <a href="#">BytesToVector2Array</a> ..... | 7  |
| <a href="#">BytesToVector3Array</a> ..... | 7  |
| <a href="#">DestroyLine</a> .....         | 7  |
| <a href="#">DestroyObject</a> .....       | 7  |
| <a href="#">DrawLine</a> .....            | 8  |
| <a href="#">DrawLine3D</a> .....          | 8  |
| <a href="#">DrawLine3DAuto</a> .....      | 9  |
| <a href="#">DrawLineViewport</a> .....    | 9  |
| <a href="#">DrawPoints</a> .....          | 9  |
| <a href="#">DrawPoints3D</a> .....        | 10 |
| <a href="#">GetLineLength</a> .....       | 10 |
| <a href="#">GetLinePoint</a> .....        | 10 |
| <a href="#">GetLinePoint01</a> .....      | 10 |
| <a href="#">GetLinePoint3D</a> .....      | 11 |
| <a href="#">GetLinePoint3D01</a> .....    | 11 |
| <a href="#">GetSegmentNumber</a> .....    | 11 |
| <a href="#">MakeLine</a> .....            | 12 |
| <a href="#">MakeCircleInLine</a> .....    | 13 |
| <a href="#">MakeCurveInLine</a> .....     | 14 |
| <a href="#">MakeEllipseInLine</a> .....   | 15 |
| <a href="#">MakeRectInLine</a> .....      | 16 |

|   |    |
|---|----|
| <a href="#">MakeSplineInLine</a> .....      | 17 |
| <a href="#">MakeTextInLine</a> .....        | 18 |
| <a href="#">MakeWireframeInLine</a> .....   | 18 |
| <a href="#">ResetTextureScale</a> .....     | 18 |
| <a href="#">SetCamera</a> .....             | 19 |
| <a href="#">SetCamera3D</a> .....           | 19 |
| <a href="#">SetColor</a> .....              | 19 |
| <a href="#">SetColors</a> .....             | 20 |
| <a href="#">SetColorsSmooth</a> .....       | 20 |
| <a href="#">SetDepth</a> .....              | 20 |
| <a href="#">SetLine</a> .....               | 20 |
| <a href="#">SetLine3D</a> .....             | 21 |
| <a href="#">SetLineDistances</a> .....      | 21 |
| <a href="#">SetLineParameters</a> .....     | 21 |
| <a href="#">SetRay</a> .....                | 22 |
| <a href="#">SetRay3D</a> .....              | 22 |
| <a href="#">SetTextureScale</a> .....       | 23 |
| <a href="#">SetVectorCamDepth</a> .....     | 23 |
| <a href="#">SetWidths</a> .....             | 23 |
| <a href="#">StopDrawingLine3DAuto</a> ..... | 24 |
| <a href="#">vectorLayer</a> .....           | 24 |
| <a href="#">vectorLayer3D</a> .....         | 24 |
| <a href="#">ZeroPointsInLine</a> .....      | 24 |

## VectorManager functions and variables

|   |    |
|---|----|
| <a href="#">GetBrightnessValue</a> .....      | 25 |
| <a href="#">ObjectSetup</a> .....             | 25 |
| <a href="#">SetBrightnessParameters</a> ..... | 26 |
| <a href="#">useDrawLine3D</a> .....           | 26 |

### A note about parameter conventions

All parameters where only the type is listed must be supplied. All parameters where a default value is listed are optional, and will use the default value if they are omitted.

```
VectorLine (name : String,  
            points : Vector2[] or Vector3[],  
            material : Material,  
            lineWidth : float,  
            lineType : LineType = LineType.Discrete,  
            joins : Joins = Joins.None) : VectorLine
```

Constructs a VectorLine object with the supplied parameters. The parameters after **lineWidth** are optional and have the default values indicated, though if you supply joins, you must also supply lineType.

**name** is a string that's used to name the mesh created for the vector line. It's also used in the name of the GameObject that this constructor generates, where the complete name is "Vector " plus the supplied name. The name is also used to identify bounds meshes made with VectorManager.ObjectSetup.

**points** is a Vector2 or Vector3 array, where each entry is a point in the line using screen-space coordinates (in the case of a Vector2 array), or world units (in the case of a Vector3 array).

The line is drawn using the material supplied by **material**. If different line depths are used, the material should use a shader that writes to the depth buffer for this to work reliably. The shader should use vertex colors in order for line segment colors to work. **Note:** if null is passed for the material, a default material is used. This material uses a basic shader which works with vertex colors and line depths, but has no texture.

The **lineWidth** is the width in pixels. This is a float, so values like 1.5 are acceptable.

**lineType** is either LineType.Discrete or LineType.Continuous. For discrete lines, each line segment is made from two entries in the Vector2 array. For continuous lines, the line starts at entry 0, and each segment is continuously connected to the next point until the end is reached.

**joins** is one of Joins.None, Joins.Fill, or Joins.Weld. Joins.None is the default and draws line segments as standard rectangles, primarily used with thin lines. Joins.Fill will fill in the gaps seen where lines join at an angle, primarily used with thick lines with no texture. This only works if using LineType.Continuous. Joins.Weld is similar, except vertices of sequential line segments are welded, which makes it more appropriate for textured lines. This works with LineType.Discrete, but only for sequential line segments.

```
VectorLine (name : String,  
            points : Vector2[] or Vector3[],  
            color : Color,  
            material : Material,  
            lineWidth : float,  
            lineType : LineType = LineType.Discrete,  
            joins : Joins = Joins.Open) : VectorLine
```

As above, but the supplied **color** creates a Color array of the appropriate length for the **lineType** (half the length of the **points** array for Discrete, the length minus one for Continuous), filled with **color**. The line will be drawn with this color as long as the shader used in the material for the line works with vertex colors.

```
VectorLine (name : String,  
            points : Vector2[] or Vector3[],  
            color : Color[],  
            material : Material,  
            lineWidth : float,  
            lineType : LineType = LineType.Discrete,  
            joins : Joins = Joins.Open) : VectorLine
```

As above, but the supplied **color** is a Color array where each entry describes a color for a line segment in the **points** array. This Color array must be half the length of the **points** array for LineType.Discrete (since each line segment is created from two points), and the length minus one for LineType.Continuous. In other words, one entry in the Color array for each line segment.

## VectorLine.active

```
var active : boolean;
```

Sets a VectorLine active or inactive. Inactive VectorLines have their renderer disabled, and if an inactive VectorLine is used with functions like DrawLine, the function will return immediately and do nothing.

## VectorLine.capLength

```
var capLength : float;
```

Sets the line's **capLength** to the supplied float. This is the number of pixels added to either end of the line (0 by default). Typically used for filling in gaps seen in thick lines when they are joined at right angles, such as when drawing selection boxes or other rectangles.

## VectorLine.continuous

```
var continuous : boolean; (read only)
```

True if the VectorLine was made with LineType.Continuous, false if it was made with LineType.Discrete.

## VectorLine.depth

```
var depth : int;
```

Sets the line's depth to the supplied integer value. It will be clamped between 0 and 100. This is used to specify the order in which lines are drawn, primarily used to determine which lines are drawn on top in the case when they overlap. Lines with higher values are drawn on top of lines with lower values, as long as the lines are drawn with materials that use shaders with ZWrite On, so they write to the zbuffer. Vector.DrawLine must be called after setting the lineDepth value for the effect to be visible on-screen.

## VectorLine.layer

```
var layer : int;
```

Sets the line object's layer to the supplied integer value. Layers are used with camera culling masks. Lines by default use layer 31, unless drawn with DrawLine3D, in which case they use layer 0 by default.

## VectorLine.lineWidth

```
var lineWidth : float;
```

The width of the line, in pixels. This can be used to change the width of a VectorLine object after it's been declared. If a VectorLine has multiple widths, then setting lineWidth will set all widths in the array to the supplied value, and reading lineWidth will return the value of the first entry in the array. Vector.DrawLine must be called after setting the lineWidth value for the effect to be visible on-screen.

## VectorLine.maxDrawIndex

```
var maxDrawIndex : int;
```

When drawn, the line drawing routine will stop with this index in the Vector2 or Vector3 array. This is 0 by default, which means the maximum index value of the Vector2 or Vector3 array, and therefore is the same as the array's length-1. Used for updating part of the line instead of the whole thing, which can be used for optimization. If the line has been drawn previously, the segments after minDrawIndex are untouched, rather than being erased. Also affects updating of line segment colors with Vector.SetColors.

## VectorLine.minDrawIndex

```
var minDrawIndex : int;
```

When drawn, the line drawing routine will start with this index in the Vector2 or Vector3 array. This is 0 by default. Used for updating part of the line instead of the whole thing, which can be used for optimization. If the line has been drawn previously, the segments before minDrawIndex are untouched, rather than being erased. Also affects updating of line segment colors with Vector.SetColors.

## VectorLine.name

```
var minDrawIndex : string;
```

Returns the name of the line that was supplied when the VectorLine was constructed. If the name is changed using VectorLine.name after the line has been constructed, then the VectorLine.vectorObject and mesh names will also change appropriately.

## VectorLine.smoothWidth

```
var smoothWidth : boolean;
```

Should line segment widths in this VectorLine be smoothly interpolated between segments? By default this is false, so each segment has its own discrete width. Line segment widths are set with [Vector.SetWidths](#).

## VectorLine.Resize

```
static function Resize (linePoints : Vector2[]) : void
```

Resizes the Vector2 points array in this VectorLine to that supplied by **linePoints**. Vector.DrawLine must be called afterwards for the new line to show up. If line segment widths have been supplied with Vector.SetWidths, they must be reset, and the same applies to Vector.SetColors (if a color array has been used for the line, all entries will be initially set to whatever the first color in the color array was).

```
static function Resize (linePoints : Vector3[]) : void
```

As above, but for lines using a Vector3 array.

```
static function Resize (newSize : int) : void
```

Resizes the number of points for the Vector2 or Vector3 array in this VectorLine to the number supplied by **newSize**. The points in the newly sized array are all set to zero, and the reference to the original Vector2 or Vector3 array used for making this VectorLine will no longer be valid, so VectorLine.points2 or VectorLine.points3 should be used instead. In other words, here's a line set to 4 points first, and resized to 2 points:

```
var linePoints = [Vector2(10, 10), Vector2(50, 50), Vector2(100, 100), Vector2(200, 200)];
var line = new VectorLine("Example", linePoints, null, 2.0);
line.Resize(2);
linePoints = line.points2;
linePoints[0] = Vector2(20, 20);
linePoints[1] = Vector2(150, 150);
Vector.DrawLine(line);
```

```
VectorPoints (name : String,  
               points : Vector2[] or Vector3[],  
               material : Material,  
               width : float) : VectorPoints
```

Constructs a VectorPoints object for a Vector2 array with the supplied parameters. This is used when calling Vector.DrawPoints, rather than a VectorLine object.

**name** is a string that's used to name the mesh created for the vector points. It's also used in the name of the GameObject that this constructor generates, where the complete name is "Vector " plus the supplied name.

**points** is a Vector2 or Vector3 array, where each entry is a point using screen-space coordinates (for Vector2 arrays) or world-space coordinates (for Vector3 arrays).

The points are drawn using the material supplied by **material**. If different depths are used, the material should use a shader that writes to the depth buffer for this to work reliably. The shader should use vertex colors for point colors to work. If null is passed for the material, a default material is used. This material uses a basic shader which works with vertex colors and line depths, but has no texture.

The **width** is the width in pixels. This is a float, so values like 1.5 are acceptable.

Most functions that accept a VectorLine object will also work with VectorPoints. Most of the variables for VectorLine (depth, minDrawIndex, etc.) work with VectorPoints, except for capLength and smoothWidth.

```
VectorPoints (name : String,  
               points : Vector2[],  
               color : Color,  
               material : Material,  
               width : float) : VectorPoints
```

As above, but the supplied **color** creates a Color array of the appropriate length, filled with **color**. The points will be drawn with this color as long as the shader used in the material for the line works with vertex colors.

```
VectorPoints (name : String,  
               points : Vector2[],  
               color : Color[],  
               material : Material,  
               width : float) : VectorPoints
```

As above, but the supplied **color** is a Color array where each entry describes a color for the corresponding point in the **points** array. All points will therefore be drawn with the respective color, as long as the shader used in the material for the line works with vertex colors.

All of the variables for VectorLine can be used with VectorPoints as well.

## BytesToVector2Array

```
static function BytesToVector2Array (lineBytes : byte[]) : Vector2
```

Converts the bytes from TextAsset.bytes to a Vector2 array, used for making specific lines without having to use long strings of hard-coded Vector2 array data in scripts. These TextAssets are made with the LineMaker editor script.

## BytesToVector3Array

```
static function BytesToVector3Array (lineBytes : byte[]) : Vector3
```

Converts the bytes from TextAsset.bytes to a Vector3 array, used for making specific lines without having to use long strings of hard-coded Vector3 array data in scripts. These TextAssets are made with the LineMaker editor script.

## DestroyLine

```
static function DestroyLine (ref line : VectorLine) : void
```

Removes a VectorLine and all associated Unity objects from the scene. If **line** is null, it's ignored and no null reference exception errors are possible.

## DestroyObject

```
static function DestroyObject (ref line : VectorLine,  
                                gameObject : GameObject) : void
```

Removes a VectorLine and all associated Unity objects from the scene, and destroys **gameObject** at the same time. If **line** or **gameObject** are null, they are ignored and no null reference exception errors are possible.

## DrawLine

```
static function DrawLine (line : VectorLine) : void
```

Draws the VectorLine object **line** on the screen. If Vector.SetCamera has not been called, it's called the first time DrawLine is used, using the default parameters. Lines with either Vector2 or Vector3 arrays can be used.

```
static function DrawLine (line : VectorLine,  
                          transform : Transform) : void
```

Draws the VectorLine object **line** on the screen, with the matrix from **transform** applied. If Vector.SetCamera has not been called, it's called the first time DrawLine is used, using the default parameters. Lines with either Vector2 or Vector3 arrays can be used.

## DrawLine3D

```
static function DrawLine3D (line : VectorLine) : void
```

Draws the VectorLine object **line** on the screen in 3D space (in contrast to DrawLine, which draws all lines in 2D with a separate camera overlaid on top of the standard camera). The array of points used to create the VectorLine object must be of type Vector3[]. If Vector.SetCamera3D has not been called, it's called the first time DrawLine is used, using the default parameters. Since the lines exist in the scene and are not drawn in a separate overlay as they are with DrawLine, they must be updated when the camera used in SetCamera changes position or else they will no longer have the correct appearance.

DrawLine3D doesn't use the vector camera object as created by SetCamera. Lines drawn with DrawLine3D are put on layer 0 by default. This default layer can be changed by setting [Vector.vectorLayer3D](#).

```
static function DrawLine3D (line : VectorLine,  
                          transform : Transform) : void
```

Draws the VectorLine object **line** on the screen in 3D space, with the matrix from **transform** applied. Otherwise the behavior is the same as DrawLine3D without the transform parameter.



## DrawLine3DAuto

```
static function DrawLine3DAuto (line : VectorLine,  
                                time : float = Mathf.Infinity) : void
```

Draws the VectorLine object **line** as DrawLine3D does, but automatically accounts for camera movement of the camera set with Vector.SetCamera. Therefore, lines drawn with DrawLine3DAuto don't need to be updated manually and will always appear correct.

The optional **time** is the number of seconds for which the line will be drawn, after which it's destroyed. The default is that the line is never destroyed.

```
static function DrawLine3DAuto (line : VectorLine,  
                                time : float = Mathf.Infinity,  
                                transform : Transform) : void
```

As above, but with the matrix from **transform** applied.

## DrawLineViewpoint

```
static function DrawLineViewport (line : VectorLine) : void
```

Draws the VectorLine object **line** on the screen. If Vector.SetCamera has not been called, it's called the first time DrawLine is used, using the default parameters. Only lines with Vector2 arrays can be used. The coordinates used in the Vector2 array are normalized viewport coordinates, such as those used for GUILayout and GUILayout objects, where (0, 0) is the lower-left corner and (1, 1) is the upper-right corner, regardless of resolution.

```
static function DrawLineViewport (line : VectorLine,  
                                  transform : Transform) : void
```

As above, but with the matrix from **transform** applied.

## DrawPoints

```
static function DrawPoints (points : VectorPoints) : void
```

Draws points rather than lines. The **points** must be created using VectorPoints rather than VectorLine.

```
static function DrawPoints (points : VectorPoints,  
                            transform : Transform) : void
```

Draws points rather than lines, with the matrix from **transform** applied. Otherwise the behavior is the same as DrawPoints without the transform parameter.

## DrawPoints3D

```
static function DrawPoints3D (points : VectorPoints) : void
```

Same as [DrawPoints](#), except the points are drawn in the scene instead of on top of it, and the points must be created with a Vector3 array. Since the points exist in the scene and are not drawn in a separate overlay as they are with DrawPoints, they must be updated when the camera used in SetCamera changes position or else they will no longer have the correct appearance.

```
static function DrawPoints3D (points : VectorPoints,  
                             transform : Transform) : void
```

Same as [DrawPoints](#) with a transform parameter, except the points are drawn in the scene instead of on top of it, and the points must be created with a Vector3 array.

## GetLineLength

```
static function GetLineLength (line : VectorLine) : float
```

Returns the total length of all line segments that make up **line**. The length refers to pixels when used with Vector2 lines, and world units when used with Vector3 lines. If the points that make up the line array are changed after GetLineLength is called, then [SetLineDistances](#) should be called before using GetLineLength again in order to maintain accurate segment distances.

## GetLinePoint

```
static function GetLinePoint (line : VectorLine,  
                             distance : float) : Vector2
```

Returns a Vector2 in screen-space coordinates that corresponds to the given **distance** in **line**. If distance is equal to or less than 0, then the first point in **line** is returned. If the distance is equal to or greater than the total length of the line as defined by [GetLineLength](#), then the last point in **line** is returned. If the points that make up the line array are changed after GetLinePoint is called, then [SetLineDistances](#) should be called before using GetLinePoint again in order to maintain accurate segment distances.

## GetLinePoint01

```
static function GetLinePoint01 (line : VectorLine,  
                               distance : float) : Vector2
```

Same as [GetLinePoint](#), except that **distance** is normalized between 0.0 and 1.0, where 0.0 is the first point in **line** and 1.0 is the last point in **line**.

## GetLinePoint3D

```
static function GetLinePoint3D (line : VectorLine,  
                                distance : float) : Vector3
```

Returns a Vector3 in world-unit coordinates that corresponds to the given **distance** in **line**. If distance is equal to or less than 0, then the first point in **line** is returned. If the distance is equal to or greater than the total length of the line as defined by [GetLineLength](#), then the last point in **line** is returned. If the points that make up the line array are changed after GetLinePoint3D is called, then [SetLineDistances](#) should be called before using GetLinePoint3D again in order to maintain accurate segment distances.

## GetLinePoint3D01

```
static function GetLinePoint3D01 (line : VectorLine,  
                                   distance : float) : Vector3
```

Same as [GetLinePoint3D](#), except that **distance** is normalized between 0.0 and 1.0, where 0.0 is the first point in **line** and 1.0 is the last point in **line**.

## GetSegmentNumber

```
static function GetSegmentNumber (line : VectorLine) : int
```

Returns the number of line segments that it would be possible to make with the given **line**. This is the number of points in the Vector2 or Vector3 array minus one for continuous lines, and half the number of points for discrete lines.

## MakeLine

```
static function MakeLine (name : String,  
                           points : Vector2[] or Vector3[]) : VectorLine
```

Creates a VectorLine from the parameters given in [Vector.SetLineParameters](#), using the supplied **name** and Vector2 or Vector3 array in **points**. This can be used as a shortcut, instead of supplying all parameters when constructing a VectorLine. If SetLineParameters has not been called first, an error will be generated.

```
static function MakeLine (name : String,  
                           points : Vector2[] or Vector3[],  
                           color : Color) : VectorLine
```

Creates a VectorLine from the parameters given in Vector.SetLineParameters, using the supplied **name** and Vector2 or Vector3 array in **points**, and overrides the default color with the supplied **color**.

```
static function MakeLine (name : String,  
                           points : Vector2[] or Vector3[],  
                           color : Color[]) : VectorLine
```

Creates a VectorLine from the parameters given in Vector.SetLineParameters, using the supplied **name** and Vector2 or Vector3 array in **points**, and overrides the default color with the supplied **color** array.

## MakeCircleInLine

```
static function MakeCircleInLine (line : VectorLine,  
                                origin : Vector2 or Vector3,  
                                up : Vector3 = Vector3.forward,  
                                radius : float,  
                                segments : int,  
                                pointRotation : float = 0.0,  
                                index : int = 0) : void
```

Creates a circle in the Vector2 or Vector3 array for VectorLine **line**. If using Vector2, the supplied **origin** is in screen space pixels, as is the supplied **radius**. If using Vector3, the coordinates are world space. The supplied **segments** indicates how many line segments will be used to create the circle, with a minimum of 3.

The optional **up** vector indicates the orientation of circles when drawn using Vector3 arrays. It has no effect when using a Vector2 array. The default of Vector3.forward means that the circle is drawn in the X/Y plane. Using Vector3.up would mean that the circle would be drawn in the X/Z plane. The up vector can be any arbitrary vector and does not need to be normalized.

The optional **pointRotation** describes how many degrees clockwise the points will be rotated around the origin. Negative values rotate the points counter-clockwise.

The optional **index** is 0 by default, though it can be anything, as long as the number of segments would fit in the Vector2 or Vector3 array. This allows creation of multiple circles in the same line, since the points used to create the circle start at the value defined by **index**. The length of the Vector2 or Vector3 array used for **line** must be at least the number of **segments** specified plus one if the line was created as continuous line, or twice the number of **segments** if it was created as a discrete line.

```
static function MakeCircleInLine (line : VectorLine,  
                                origin : Vector2 or Vector3,  
                                radius : float) : void
```

As above, but since the number of segments isn't specified, the maximum allowed by the entire Vector2 or Vector3 array will be used.

## MakeCurveInLine

```
static function MakeCurveInLine (line : VectorLine,
                                curvePoints : Vector2[] or Vector3[],
                                segments : int,
                                index : int = 0) : void
```

Creates a bezier curve in the Vector2 or Vector3 array for VectorLine **line**. The supplied **curvePoints** is a Vector2 or Vector3 array that must contain four elements, where the elements are Vector2s using screen space pixel coordinates or Vector3s using world space coordinates, and are defined as follows:

curvePoints[0] = the first anchor point of the curve  
 curvePoints[1] = the first control point of the curve  
 curvePoints[2] = the second anchor point of the curve  
 curvePoints[3] = the second control point of the curve

The supplied **segments** indicates how many line segments will be used to create the curve, with a minimum of 2. The optional **index** is 0 by default, though it can be anything, as long as the number of segments would fit in the **line** array. This allows creation of multiple curves in the same line, since the points used to create the curve start at the value defined by **index**. The length of the array used for **line** must be at least the number of **segments** specified plus one if the line was created as continuous line, or twice the number of **segments** if it was created as a discrete line.

Example: a curve with 10 segments would require an array in **line** that contains 11 points, as long as **line** is a continuous line. If it was a discrete line, the array would need to contain 20 points. Two curves of 10 segments each in the same line would require double the number of points, or 22 and 40 respectively. The index for the second curve would be 11 for a continuous line or 20 for a discrete line.

```
static function MakeCurveInLine (line : VectorLine,
                                anchor1 : Vector2 or Vector3,
                                control1 : Vector2 or Vector3,
                                anchor2 : Vector2 or Vector3,
                                control2 : Vector2 or Vector3,
                                segments : int,
                                index : int = 0) : void
```

The anchor and control points for the curve are defined as individual Vector2s or Vector3s rather than a Vector2[] or Vector3[] array, but otherwise this is the same as above. The Vector2s use screen space pixel coordinates, and Vector3s use world space coordinates.

```
static function MakeCurveInLine (line : VectorLine,
                                curvePoints : Vector2[] or Vector3[] : void
```

As above, but since the number of segments isn't specified, the maximum allowed by the entire Vector2 or Vector3 array will be used.

## MakeEllipseInLine

```
static function MakeEllipseInLine (line : VectorLine,  
                                   origin : Vector2 or Vector3,  
                                   up : Vector3 = Vector3.forward,  
                                   xRadius : float,  
                                   yRadius : float,  
                                   segments : int,  
                                   pointRotation : float = 0.0,  
                                   index : int = 0) : void
```

Creates an ellipse in the Vector2 or Vector3 array for VectorLine **line**. For Vector2, the supplied **origin** is in screen space pixels, as are the supplied radii. Vector3 uses world space coordinates. **xRadius** is the horizontal radius of the ellipse, and **yRadius** is the vertical radius. The supplied **segments** indicates how many line segments will be used to create the ellipse, with a minimum of 3.

The optional **up** vector indicates the orientation of ellipses when drawn using Vector3 arrays. It has no effect when using a Vector2 array. The default of Vector3.forward means that the ellipse is drawn in the X/Y plane. Using Vector3.up would mean that the ellipse would be drawn in the X/Z plane. The up vector can be any arbitrary vector and does not need to be normalized.

The optional **pointRotation** describes how many degrees clockwise the points will be rotated around the origin. Negative values rotate the points counter-clockwise.

The optional **index** is 0 by default, though it can be anything, as long as the number of segments would fit in the Vector2 or Vector3 array. This allows creation of multiple ellipses in the same line, since the points used to create the ellipse start at the value defined by **index**. The length of the Vector2 or Vector3 array used for **line** must be at least the number of **segments** specified plus one if the line was created as continuous line, or twice the number of **segments** if it was created as a discrete line.

```
static function MakeEllipseInLine (line : VectorLine,  
                                   origin : Vector2 or Vector3,  
                                   xRadius : float,  
                                   yRadius : float) : void
```

As above, but since the number of segments isn't specified, the maximum allowed by the entire Vector2 or Vector3 array will be used.

## MakeRectInLine

```
static function MakeRectInLine (line : VectorLine,  
                                rect : Rect,  
                                index : int = 0) : void
```

Creates a rectangle in the Vector2 or Vector3 array for VectorLine **line**. The supplied **rect** is in screen space pixels if using Vector2, or world space coordinates if using Vector3.

The optional **index** is 0 by default, though it can be anything, as long as the rect would fit in the Vector2 or Vector3 array. This allows creation of multiple rectangles in the same line, since the points used to create the rectangle start at the value defined by **index**. The length of the Vector2 or Vector3 array used for **line** must be at least 5 if the line was created as continuous line, or 8 if it was created as a discrete line.

```
static function MakeRectInLine (line : VectorLine,  
                                topLeft : Vector2 or Vector3,  
                                topRight : Vector2 or Vector3,  
                                index : int = 0) : void
```

Creates a rectangle in the Vector2 or Vector3 array for VectorLine **line**. The supplied **topLeft** and **topRight** describe the respective corners of the rectangle in screen space pixels if using Vector2, or world space coordinates if using Vector3. Otherwise this is the same as MakeRectInLine using a Rect.



## MakeSplineInLine

```
static function MakeSplineInLine (line : VectorLine,  
                                   splinePoints : Vector2[] or Vector3[],  
                                   segments : int,  
                                   index : int = 0,  
                                   loop : boolean = false) : void
```

Creates a Catmull-Rom spline in the Vector2 or Vector3 array for VectorLine **line**. The supplied **splinePoints** are a Vector2 (screen space) or Vector3 (world space) array defining the points that should be used to create the spline. The resulting curve will pass directly through all points in the splinePoints array. The supplied **segments** indicates how many line segments will be used to create the circle, with a minimum of 3. **loop** indicates whether the first and last points in the splinePoints array will be connected or not.

The optional **index** is 0 by default, though it can be anything, as long as the number of segments would fit in the Vector2 or Vector3 array. This allows creation of multiple splines in the same line, since the points used to create the spline start at the value defined by **index**. The length of the Vector2 or Vector3 array used for **line** must be at least the number of **segments** specified plus one if the line was created as continuous line, or twice the number of **segments** if it was created as a discrete line.

```
static function MakeSplineInLine (line : VectorLine,  
                                   splinePoints : Vector2[] or Vector3[],  
                                   loop : boolean = false) : void
```

As above, but since the number of segments isn't specified, the maximum allowed by the entire Vector2 or Vector3 array will be used.

## MakeTextInLine

```
static function MakeTextInLine (line : VectorLine,  
                                text : String,  
                                position : Vector2 or Vector3,  
                                size : float,  
                                characterSpacing : float = 1.0,  
                                lineSpacing : float = 1.5,  
                                uppercaseOnly : boolean = true) : void
```

Creates the string **text** in VectorLine **line**. The text is placed at **position** using screen-space pixel coordinates if used with Vector2 points, or world space coordinates if used with Vector3 points. **size** is the number of pixels in height if used with Vector2 points, or world units if used with Vector3 points. Text is always monospaced. Any characters not present in the default font are ignored. “\n” can be used as a newline character. If the points array for **line** is not large enough to contain the line segments that make up the text, it will be resized appropriately. If this happens, any variables referencing the original points array will no longer reference the new points array. If a new reference is needed, it should be assigned to line.points2 if the line was made with Vector2 points, or line.points3 if the line was made with Vector3 points.

The optional **characterSpacing** is 1.0 by default, which is a relative value, where 1.0 equals **size**, .5 would be half of **size**, etc. The optional **lineSpacing** is 1.5 by default, and is also a relative value in the same way. If supplying either the character spacing or the line spacing, both values must be supplied.

The optional **uppercaseOnly** is true by default, which makes the text always display using uppercase characters, even if **text** contains lowercase characters.

## MakeWireframeInLine

```
static function MakeWireframeInLine (line : VectorLine,  
                                      mesh : Mesh) : void
```

Creates a wireframe in **line** using all the triangles in **mesh**. The points array used to create **line** must be of type Vector3, and the line must be created using LineType.Discrete. If the points array for **line** is not large enough to contain the line segments that make up the wireframe, it will be resized appropriately.

## ResetTextureScale

```
static function ResetTextureScale (line : VectorLine) : void
```

Sets all UVs in the mesh object for **line** to their initial values. Used after [Vector.SetTextureScale](#) has been called, in case a return to the default state is desired.

## SetCamera

```
static function SetCamera (camera : Camera = Camera.main,  
                           clearFlags : CameraClearFlags = CameraClearFlags.DepthOnly,  
                           useOrtho : boolean = false) : Camera
```

Sets the camera up for line drawing. This generally has to be done once at startup, though it also needs to be called after any screen resolution changes, and may need to be called after level changes, if any non-vector cameras aren't manually set to ignore layer 31.

This function is used automatically with the default parameters the first time `Vector.DrawLine`, `Vector.DrawPoints`, `Vector.SetLine`, or `Vector.SetRay` is called. Therefore it only needs to be called manually if parameters other than the defaults are desired.

The optional non-vector **camera** is set to the first camera in the scene found tagged "Main Camera" by default. This camera is used for supplying the viewpoint for any 3D vector objects, and has its culling mask set to ignore layer 31, so lines aren't visible except to the vector camera. Setting [Vector.vectorLayer](#) to an integer, before calling `SetCamera`, will cause the line-drawing layer to be that supplied rather than 31.

The optional **clearFlags** for the vector camera is `DepthOnly` by default. Passing **clearFlags** will change this to the supplied value instead.

The optional **useOrtho** is `false` by default. Passing `true` will cause the vector camera to use orthographic mode, which may render lines slightly more accurately, but can potentially cause anomalies in 3D lines under certain circumstances.

`SetCamera` returns the vector camera that's created. This can be used if further modifications to the vector camera are desired.

## SetCamera3D

```
static function SetCamera3D (camera : Camera = Camera.main) : void
```

Sets the camera up for drawing lines with `Vector.DrawLine3D`. The optional **camera** is set to the first camera in the scene found tagged "Main Camera" by default. In contrast to `Vector.SetCamera`, the culling mask for the supplied camera is not altered, and the vector camera normally used for drawing lines is not created. This function is called automatically when `DrawLine3D` is used for the first time, so it's normally not necessary to call it manually, unless using a camera other than "Main Camera" is desired.

## SetColor

```
static function SetColor (line : VectorLine,  
                          color : Color) : void
```

Sets all the line segment colors in **line** to the supplied **color**. The line has its color changed immediately without having to call `Vector.DrawLine`.

## SetColor

```
static function SetColor (line : VectorLine,  
                           color : Color[]) : void
```

Sets all the line segment colors in **line** to the supplied **color** array. The line has its colors changed immediately without having to call `Vector.DrawLine`. Each entry in the color array corresponds to a line segment, so the length of the color array must be half the length of the `Vector2` or `Vector3` array in **line** if using a discrete line, or the length of the `Vector2` or `Vector3` array minus one if using a continuous line.

## SetColorSmooth

```
static function SetColorSmooth (line : VectorLine,  
                                  color : Color[]) : void
```

Sets all the line segment colors in **line** to the supplied color array, as with `SetColor`. Additionally, vertex colors are smoothly blended. The line has its colors changed immediately without having to call `Vector.DrawLine`.

## SetDepth

```
static function SetDepth (transform : Transform,  
                           depth : int) : void
```

Sets the line depth of **transform** to the value supplied by **depth**. This can be used for special effects, where transforms (usually planes) are positioned in the space used by vector lines. The object must be drawn on the same layer as lines for it to be visible to the vector camera. The depth for `VectorLines` is set with [VectorLine.depth](#).

## SetLine

```
static function SetLine (color : Color,  
                          time : float = Mathf.Infinity,  
                          params points : Vector2[] or Vector3[]) : VectorLine
```

Creates a `VectorLine` using the supplied **points**, and draws it immediately using the supplied **color**. The points use screen-space coordinates if made with `Vector2`s, or world coordinates if made with `Vector3`s. “Params” means that each point is supplied individually, rather than as an array. At least two `Vector2`s are required; this will create a single line segment. Each additional `Vector2` will extend the line to that point by adding another line segment from the last. `SetLine` returns a `VectorLine` object, so it can be assigned to a variable and used in any function that takes a `VectorLine`.

The optional **time** parameter specifies how long, in seconds, the line will be drawn before it’s destroyed. If this is left out, the line will never be removed.

If `Vector.SetCamera` has not been called, it’s called the first time `SetLine` is used, using the default parameters.

## SetLine3D

```
static function SetLine3D (color : Color,  
                           params points : Vector3[]) : VectorLine
```

Creates a VectorLine using the supplied **points**, and draws it immediately using the supplied **color**. The points use world-space coordinates. “Params” means that each point is supplied individually, rather than as an array. At least two Vector3s are required; this will create a single line segment. Each additional Vector3 will extend the line to that point by adding another line segment from the last. SetLine3D returns a VectorLine object, so it can be assigned to a variable and used in any function that takes a VectorLine.

The lines created by SetLine3D are drawn in world space, rather than on top of other objects, and are not seen by the vector camera, which is not required for 3D lines.

If Vector.SetCamera3D has not been called, it's called the first time SetLine is used, using the default parameters.

## SetLineDistances

```
static function SetLineDistances (line : VectorLine) : void
```

Used to recompute line segment distances in **line** after changing any of the Vector2 or Vector3 points that make up the line. If [GetLineLength](#) or any of the [GetLinePoint](#) functions have been used, and the line points are changed, those functions will no longer return accurate information unless SetLineDistances is called before using those functions again. It's not necessary to call SetLineDistances before using GetLineLength or any of the GetLinePoint functions for the first time.

## SetLineParameters

```
static function SetLineParameters (color : Color,  
                                     material : Material,  
                                     lineWidth : float,  
                                     capLength : float,  
                                     depth : int,  
                                     lineType : LineType,  
                                     joins : Joins) : void
```

Used to set up the default parameters for [Vector.MakeLine](#). The color, material, line width, end cap length, depth, LineType, and Joins are then used whenever constructing a line with Vector.MakeLine. When this is called again, any further lines will then use the new defaults, but already-created lines will be unaffected.

## SetRay

```
static function SetRay (color : Color,  
                        time : float = Mathf.Infinity,  
                        origin : Vector3,  
                        direction : Vector3) : VectorLine
```

Creates a VectorLine from **origin** to **origin** + **direction**, and draws it immediately using the supplied **color**. SetRay returns a VectorLine object, so it can be assigned to a variable and used in any function that takes a VectorLine.

The optional **time** parameter specifies how long, in seconds, the line will be drawn before it's destroyed. By default, the line will never be removed.

If Vector.SetCamera has not been called, it's called the first time SetRay is used, using the default parameters.

## SetRay3D

```
static function SetRay3D (color : Color,  
                           time : float = Mathf.Infinity,  
                           origin : Vector3,  
                           direction : Vector3) : VectorLine
```

Creates a VectorLine from **origin** to **origin** + **direction**, and draws it immediately using the supplied **color**. The line is drawn in world space, rather than on top of other objects, and is not seen by the vector camera, which is not required for 3D lines. SetRay3D returns a VectorLine object, so it can be assigned to a variable and used in any function that takes a VectorLine.

The optional **time** parameter specifies how long, in seconds, the line will be drawn before it's destroyed. If this is left out, the line will never be removed.

If Vector.SetCamera3D has not been called, it's called the first time SetRay is used, using the default parameters.

## SetTextureScale

```
static function SetTextureScale (line : VectorLine,  
                                textureScale : float,  
                                offset : float = 0.0) : void
```

Changes the UV mapping for **line**, so that the texture, instead of being stretched to fill a line segment, is uniformly scaled such that its width in the direction of the line segment is **textureScale** times that of its height. Using 1.0 for square textures results in the width and height being the same. How many times the texture repeats along the line segment therefore depends on the length of the line segment. The optional **offset** value offsets the texture along the line a certain percentage of the scale, with 1.0 being 100%. This is the same visual effect as altering the material's `mainTextureScale.x` value, though the material in this case is not affected. Once `Vector.SetTextureScale` has been used, `Vector.ResetTextureScale` can be used to reset the line back to its original appearance.

```
static function SetTextureScale (line : VectorLine,  
                                transform : Transform,  
                                textureScale : float,  
                                offset : float = 0.0) : void
```

As above, but accounts for the position and rotation of the **transform** that's passed in. Use this to set the scale properly for any lines that are drawn with `DrawLine` or `DrawLine3D`, using the same transform.

## SetVectorCamDepth

```
static function SetVectorCamDepth (depth : int) : void
```

Sets the depth of the vector cam used to show lines drawn with `Vector.DrawLines`. By default, when `Vector.SetCamera` is called, the depth is one greater than the depth of the standard camera, so that lines are drawn on top of the view. By setting **depth** to a different value, the depth of the vector camera can be changed, which is normally only done for special effects.

## SetWidths

```
static function SetWidths (line : VectorLine,  
                           lineWidths : float[] or int[]) : void
```

Sets the pixel widths of the line segments in **line** to the values supplied by the **lineWidths** array, which can be either a float array or an int array. Each entry in the line widths array corresponds to a line segment, so the length of the line widths array must be half the length of the `Vector2` or `Vector3` array in **line** if using a discrete line, or the length of the `Vector2` or `Vector3` array minus one if using a continuous line. `Vector.DrawLine` must be called afterwards in order for the new widths to show up. Whether each line segment is a distinct width, or the widths are smoothly blended, is determined by [VectorLine.smoothWidth](#).

## StopDrawingLine3DAuto

```
static function StopDrawingLine3DAuto (line : VectorLine) : void
```

Stops the automatic update of any line drawn with [DrawLine3DAuto](#).

## vectorLayer

```
var vectorLayer : int = 31
```

Sets the default layer that VectorLines will be drawn in, and which the vector camera will see. Should be set before using Vector.SetCamera for it to have any effect.

## vectorLayer3D

```
var vectorLayer3D : int = 0
```

Sets the default layer that lines drawn with DrawLine3D will be drawn in.

## ZeroPointsInLine

```
static function ZeroPointsInLine (line : VectorLine,  
                                   index : int = 0) : void
```

Sets points in the Vector2 or Vector3 array in **line** to Vector2.zero or Vector3.zero respectively. By default it starts from index 0 and sets all points to zero, but an **index** value greater than 0 can be supplied, which zeroes out points starting from that index (must be less than the Vector2 or Vector3 array length).



## GetBrightnessValue

```
static function GetBrightnessValue (position : Vector3) : float
```

Given the distance of **position** from the non-vector camera used in `Vector.SetCamera`, returns a float between 0 and 1, where 0 is 0% brightness and 1 is 100% brightness.

## ObjectSetup

```
static function ObjectSetup (gameObject : GameObject,  
                             vectorLine : VectorLine,  
                             visibility : Visibility,  
                             brightness : Brightness,  
                             makeBounds : boolean = true) : void
```

Makes **gameObject** have a “shadow” 3D vector object as defined by **vectorLine**, which behaves according to the transform of **gameObject**. Depending on the values of **visibility** and **brightness**, one or more components may be attached to **gameObject** when this function is called.

**visibility** can be `Visibility.Dynamic`, `Visibility.Static`, `Visibility.Always`, or `Visibility.None`. `Visibility.Dynamic` causes the vector line to always be drawn when **gameObject** is visible to a camera, using the object’s transform. `Visibility.Static` only draws the vector line if the camera moves and the **gameObject** is visible to a camera, and is for objects that never move, since the object’s transform is only used once when the 3D vector line is initialized. `Visibility.Always` causes the vector line to be drawn every frame and has no optimizations. `Visibility.None` doesn’t add any visibility components, and removes any `Visibility` scripts if `ObjectSetup` has been called on this **gameObject** before.

**brightness** can be `Brightness.Fog` or `Brightness.Normal`. `Brightness.Fog` adds the `BrightnessControl` component, which makes the 3D vector object’s color behave according to the parameters given in [VectorManager.SetBrightnessParameters](#). If `SetBrightnessParameters` hasn’t been called, the defaults for that function will be used. Currently only the first color in the array is used for the entire object. `Brightness.None` doesn’t add any component, so the vector line’s color won’t be altered, and removes `BrightnessControl` if `ObjectSetup` has been called on this **gameObject** before.

The optional **makeBounds** by default creates an invisible bounds mesh for the **gameObject**’s mesh filter, so that `OnBecameInvisible` and `OnBecameVisible` will still work, which allows optimizations for `Visibility.Dynamic` and `Visibility.Static`. If set to false, then the **gameObject**’s mesh is not replaced by a bounds mesh. One bounds mesh is created for each `VectorLine` name, so `VectorLines` that share a name will also share a bounds mesh.

## SetBrightnessParameters

```
static function SetBrightnessParameters (minBrightnessDistance : float = 500,  
                                         maxBrightnessDistance : float = 250,  
                                         brightnessLevels : int = 32,  
                                         frequency : float = .2,  
                                         color : Color = Color.black) : void
```

Sets parameters for objects that use Brightness.Fog with VectorManager.ObjectSetup.

**minBrightnessDistance** is the distance that the object must be from the non-vector camera in order to have the minimum amount of brightness, or in other words have 100% “fog” color. **maxBrightnessDistance** is the distance that the object must be from the non-vector camera in order to have the maximum amount (100%) of brightness. The color of this brightness is taken from entry 0 in the vector line’s segmentColors array. The vector line will have **brightnessLevels** “steps” between 0% and 100% brightness. The fewer steps, the more obvious the transitions become as the object moves closer and farther from the camera. How often the distance of objects is checked depends on **frequency**, which by default is 5 times per second. The **color** defines the “fog” color.

## useDrawLine3D

```
var useDrawLine3D : bool = false
```

Tells the VectorManager routines to use Vector.DrawLine3D if VectorManager.useDrawLine3D is set to true, or Vector.DrawLine if set to false, which is the default. See [Vector.DrawLine3D](#) for more details about 3D lines.