

Extensions

Extensions are methods of extending many types of data in Unity, which greatly simplifies everyday tasks when creating games.

Customization

To work with extensions, simply include the `Redcode.Extensions` namespace in your code.

```
Using Redcode.Extensions;
```

In some cases, method names in this documentation will end with **XX**, **XXX** or **XXXX**, meaning you can substitute any combination of [X, Y, Z, W] (in case of vectors) or [R, G, B, A] (in case of colors) instead.

Example:

```
// changes the color of the G channel to 1f  
// and returns the result as a copy.  
color.WithG(1f);  
  
// changes color of channel R to 0.5f  
// and B to 1f and returns the result as a copy.  
color.WithRB(0.5f, 1f)  
  
// Extracts the X and Z components from the vector  
// and returns them as a Vector2 object.  
vector.GetXZ();
```

Float and Double

Remap - reassigns a number from the initial range to the number of the final range.

```
var x = 0.5f;  
var remapedX = x.Remap(0f, 1f, 4f, 6f);
```

The variable **RemapedX** will be equal to 5f.

Approximately - performs an approximate comparison of two floating-point numbers.

```
var x = 0.3333333;  
var result = x.Approximately(1f / 3f);
```

Color and Color32

With and **WithXXX** - replaces the values in the specified color channels and returns the result as a copy. Has 3 overloads.

```

var gray = Color.black.With(0, 0.5f, 1, 0.75f, 2, 1f); // R = 0.5f, G = 0.75f, B = 1f.

var green = Color.black.WithG(1f);
var pink = Color.black.WithRB(1f, 1f);
var yellow = Color.black.WithRGA(1f, 1f, 0f); // A - color alpha channel

```

Graphic

SetColorXXX - sets the value to the selected color channels of the Graphic.color property.

```

image.SetColorR(1f);
image.SetColorGB(1f, 1f);
image.SetColorRBA(0f, 0f, 0f);

```

The class Image is inherited from the class Graphic, so it has these methods just like Graphic.

Comparable

The IsBetween method checks if an object is between two other objects. Also accepts two additional parameters - boolean values indicating whether the check should be done inclusively or exclusively.

```

var x = 1;
var result1 = x.IsBetween(0, 2); // exclusively
var result2 = x.IsBetween(0, 1, true, true); // inclusively on both sides

```

IEnumerable.

GetRandomElement - returns an arbitrary sequence element.

```

var list = new List<int>() { 1, 2, 3, 4, 5 }
var element = list.GetRandomElement();

```

Except - returns a copy of the sequence without the excluded element.

```

var list = new List<int>() { 1, 2, 3, 4, 5 };
list.Except(3); // [ 1, 2, 4, 5 ]

```

Shuffled - returns a shuffled copy of the sequence.

```

var list = new List<int>() { 1, 2, 3, 4, 5 };
var shuffled = list.Shuffled();

```

AsString - Returns a string representing the elements of a sequence separated by commas and in square brackets.

```

var list = new List<int>() { 1, 2, 3, 4, 5 };
print(list.AsString());

```

IEquatable

`EqualsToAll` - compares the current object with those passed to the method, returns `true` if the `Object.Equals` method returns true for all of them. Objects passed to the method must have the same type as the original.

```
int x, y, z;  
x = y = z = 1;
```

```
var result = x.EqualsToAll(y, z);
```

`EqualsToAny` - compares the current object with the ones passed to the method, returns `true` if for at least one of them the method `Object.Equals` returns true. Objects passed to the method must be of the same type as the original.

```
int x = 1;  
int y = 1;  
int z = 2;
```

```
var result = x.EqualsToAny(y, z);
```

ICollection

`Pop` - removes the element specified by index from the list and returns it.

```
var list = new List<int>() { 1, 2, 3, 4, 5 };  
var element = list.Pop(2);
```

`PopRandom` - removes an arbitrary element from the list and returns it.

```
var list = new List<int>() { 1, 2, 3, 4, 5 };  
var element = list.PopRandom();
```

System.Object

`EqualsToAll` - compares the current object with those passed to the method, returns `true` if the `Object.Equals` method returns true for all of them.

```
int x, y, z;  
x = y = z = 1;
```

```
var result = x.EqualsToAll(y, z);
```

`EqualsToAny` - compares the current object with those passed to the method, returns `true` if for at least one of them the method `Object.Equals` returns true.

```
int x = 1;  
int y = 1;  
int z = 2;
```

```
var result = x.EqualsToAny(y, z);
```

Quaternion

`With` and `WithXXX` - replaces the values in the specified axes of the quaternion and returns the result as a copy. Has 3 overloads. Works directly with values of X, Y, Z, W quaternion (not Euler angles).

```
transform.rotation.With(0, 4f, 2, 8f, 3, -32);
```

```
transform.rotation.WithX(8f);  
transform.rotation.WithYW(32f, -16f);  
transform.rotation.WithXZW(4f, 8f, -32);
```

Rect

`WithCenter` - sets the center of the rectangle.

`WithPosition` - sets the position of the rectangle.

`WithHeight` - sets the height of the rectangle.

`WithWidth` - sets the width of the rectangle.

`WithMax` - sets the maximum point of the rectangle.

`WithMin` - sets the minimum point of the rectangle.

`WithSize` - sets the size of the rectangle.

`WithX` - sets the X position of the rectangle.

`WithY` - sets the Y position of the rectangle.

`WithXMin` - sets the minimum X position of the rectangle.

`WithYMin` - sets the minimum Y position of the rectangle.

`WithXMax` - sets the maximum X position of the rectangle.

`WithYMax` - sets the maximal Y position of the rectangle.

Example:

```
rect = rect.WithSize(Vector2.one).WithX(4f);
```

RectTransform.

`SetSizeDeltaX` and `SetSizeDeltaY` - sets one of the values of the `RectTransform.sizeDelta` property to the specified value.

```
RectTransform.SetSizeDeltaX(100f);  
RectTransform.SetSizeDeltaY(200f);
```

`SetAnchorMinX` and `SetAnchorMinY` - sets one of the values of the `RectTransform.anchorMin` property to the specified value.

```
rectTransform.SetAnchorMinX(100f);
``RectTransform.SetAnchorMinY(200f);
```

SetAnchorMaxX and SetAnchorMaxY - sets one of the values of the RectTransform.anchorMax property to the specified value.

```
rectTransform.SetAnchorMaxX(200f);
RectTransform.SetAnchorMaxY(400f);
```

SetAnchoredPositionX and SetAnchoredPositionY - sets one of the values of the RectTransform.anchoredPosition property to the specified value.

```
rectTransform.SetAnchoredPositionX(100f);
``RectTransform.SetAnchoredPositionY(100f);
```

SetAnchoredPosition3DX and SetAnchoredPosition3DXX - sets the selected value of the RectTransform.anchoredPosition3D property to the specified value.

```
rectTransform.SetAnchoredPosition3DX(50f);
The rectTransform.SetAnchoredPosition3DZ(0f);
The rectTransform.SetAnchoredPosition3DXY(10f, 20f);
```

SetPivotX and SetPivotY - sets one of the values of the RectTransform.pivot property to the specified value, with an element offset.

```
rectTransform.SetPivotX(100f);
``RectTransform.SetPivotY(100f);
```

SetPivotOnly, SetPivotOnlyX and SetPivotOnlyY - sets one of the values of the RectTransform.pivot property to the specified value, the element remains in place.

```
rectTransform.SetPivotOnlyX(100f);
The rectTransform.SetPivotOnlyY(200f);
The rectTransform.SetPivotOnly(100f, 200f);
```

GetSize - calculates and returns the actual size of the element.

```
var size = rectTransform.GetSize();
```

Scene

FindObjectsOfType - finds all objects (including those not active) in the scene with the specified component.

```
var transforms = SceneManager.GetActiveScene().FindObjectsOfType<Transform>();
```

SceneManager.

This class is an auxiliary class and does not store extension methods.

`FindObjectsOfTypeInActiveScene` - finds all objects (including those not active) in the active scene with the specified component.

```
var transforms = SceneManagerExtensions.FindObjectsOfTypeInActiveScene<Transform>();
```

`FindObjectsOfTypeInOpenScenes` - finds all objects (including inactive ones) with the specified component in all open scenes.

```
var transforms = SceneManagerExtensions.FindObjectsOfTypeInOpenScenes<Transform>();
```

Transform

`SetPositionX` and `SetPositionXX` - set the global position of the object along the selected axis.

```
transform.SetPositionX(1f);  
transform.SetPositionYZ(2f, 3f);
```

`SetLocalPositionX` and `SetLocalPositionXX` - set the local position of the object on the selected axis.

```
transform.SetLocalPositionX(1f);  
transform.SetLocalPositionYZ(2f, 3f);
```

`SetEulerAnglesX` and `SetEulerAnglesXX` - set the global rotation of the object along the selected axis.

```
transform.SetEulerAnglesX(45f);  
transform.SetEulerAnglesYZ(0f, 90f);
```

`SetLocalEulerAnglesX` and `SetLocalEulerAnglesXX` - set the local rotation of the object along the selected axis.

```
transform.SetLocalEulerAnglesX(45f);  
transform.SetLocalEulerAnglesYZ(0f, 90f);
```

`SetLocalScaleX` and `SetLocalScaleXX` - set the local scale of the object along the selected axis.

```
transform.SetLocalScaleX(2f);  
transform.SetLocalScaleXZ(2f, 3f);
```

`SetAsPreviousSibling` and `SetAsNextSibling` - shifts back or forward the current object among other game objects in the scene located in the hierarchy next to the current one.

```
rectTransform.SetAsPreviousSibling();  
rectTransform.SetAsNextSibling();
```

`GetChilds` - returns child objects of the first level.

```
var childs = transform.GetChilds();
```

`GetRandomChild` - returns random child.

```
var child = transform.GetRandomChild();
```

AddChilds - adds child objects to the current one to the end of the list.

```
transform.AddChilds(first, second, third);
```

DestroyChilds - removes all child objects.

```
transform.DestroyChilds();
```

DestroyChildsWhere - removes all child objects that satisfy the condition.

```
transform.DestroyChildsWhere(c => c.name.StartsWith("abc"));
```

DestroyChild - removes child object by index.

```
transform.DestroyChild(1);
```

DestroyFirstChild and DestroyLastChild - delete the first and last child objects respectively.

```
transform.DestroyFirstChild();
```

```
transform.DestroyLastChild();
```

GetPreviousSiblingTransform and GetNextSiblingTransform get the Transform component of the previous and next object at this level, respectively.

```
var nextObject = transform.GetNextSiblingTransform().gameObject;
```

SetActiveComponentsInChildren<T> and SetActiveComponentsInParents<T> - sets the activity of all components on this object and its children or parents respectively.

```
transform.SetActiveComponentsInChildren<BoxCollider>(false);
```

gameObject.SetActiveComponentsInChildren<T> and gameObject.SetActiveComponentsInParents<T> - sets the activity of all components on this object and its children or parents respectively.

```
gameObject.SetActiveComponentsInChildren<BoxCollider>(false);
```

Camera.

SetColor and SetColorXXX - set the color of the camera background.

```
camera.SetColorGB(1f, 0.5f);
```

SetLensShiftX and SetLensShiftY - set the camera lens shift.

SetPixelRectCenter - sets the center of the pixel coordinates of the camera on the screen.

SetPixelRectPosition - sets the position of pixel coordinates of the camera on the screen.

SetPixelRectHeight - sets the height of the pixel coordinates of the camera on the screen.

SetPixelRectWidth - sets the width of the pixel coordinates of the camera on the screen.

SetPixelRectMax - sets the position of the maximum angle of the camera pixel coordinates on the screen.

SetPixelRectMin - sets the position of the minimum angle of pixel camera coordinates on the screen.

SetPixelRectSize - sets the size of the pixel coordinates of the camera on the screen.

SetPixelRectX - sets the X position of pixel coordinates of the camera on the screen.

SetPixelRectY - sets the Y position of pixel coordinates of the camera on the screen.

SetPixelRectXMax - sets the maximum X position of pixel coordinates of the camera on the screen.

SetPixelRectYMax - sets the maximum Y position of pixel coordinates of the camera on the screen.

SetPixelRectXMin - sets the minimum X position of pixel coordinates of the camera on the screen.

SetPixelRectYMin - sets the minimum Y position of pixel coordinates of the camera on the screen.

SetRectCenter - sets the center of normalized camera coordinates on the screen.

SetRectPosition - sets the position of normalized camera coordinates on the screen.

SetRectHeight - sets the height of normalized camera coordinates on the screen.

SetRectWidth - sets the width of normalized camera coordinates on the screen.

SetRectMax - sets the position of the maximum angle of normalized camera coordinates on the screen.

SetRectMin - sets the position of the minimum angle of normalized camera coordinates on the screen.

SetRectSize - sets the size of normalized camera coordinates on the screen.

SetRectX - sets the X position of normalized camera coordinates on the screen.

SetRectY - sets the Y position of normalized camera coordinates on the screen.

SetRectXMax - sets the maximum X position of normalized camera coordinates on the screen.

SetRectYMax - sets the maximum Y position of normalized camera coordinates on the screen.

SetRectXMin - sets the minimum X position of normalized camera coordinates on the screen.

SetRectYMin - sets the minimum Y position of normalized camera coordinates on the screen.

SetSensorSizeX and **SetSensorSizeY** - sets the size of the sensor.

SetTransparencySortAxisX or **SetTransparencySortAxisXXX** - sets the vector along which the distance to objects is measured.

Vector2 and Vector2Int

With, **WithX** and **WithY** - replaces the values in the specified vector axes and returns the result as a copy.

```
Vector2.zero.With(0, 1f); // 0 is the axis number and 1f is the value on that axis.
```

```
Vector2.zero.WithX(1f);
```

```
Vector2.zero.WithY(2f);
```

GetYX - swaps the X and Y values of the vector and returns the result as a copy.

```
Vector2.up.GetYX();
```

InsertX - substitutes a value for the X position, thereby expanding the current vector to a three-dimensional vector and returning it as the result. Instead of X you can insert X, Y or Z.

```
Vector2.zero.InsertX(1f); // [1, 0, 0]
```

```
Vector2.zero.InsertY(1f); // [0, 1, 0]
```

```
Vector2.zero.InsertZ(1f); // [0, 0, 1]
```

MaxComponent and **MinComponent** - return the maximal and minimal component of the vector respectively.

```
var max = new Vector2(3.14f, 7f).MaxComponent(); // will return 7
```

```
var min = new Vector2(3.14f, 7f).MinComponent(); // will return 3.14
```

Vector3 and Vector3Int

With - replaces the values in the specified vector axes and returns the result as a copy. Has 2 overloads.

```
Vector3.zero.With(0, 1f);
```

```
Vector3.zero.With(0, 1f, 2, 3.14f);
```

WithX and **WithXX** - substitute values in the selected vector axes and return the result as a copy. Instead of X substitute X, Y or Z.

```
Vector3.zero.WithX(1f);
Vector3.zero.WithXZ(0f, 2f);
```

Get - allows you to select 2 or 3 components of the original vector and their order, thereby forming a new two- or three-dimensional vector and returning it. Has 2 overloads.

```
var vector2 = new Vector3(1f, 2f, 3f).Get(0, 2); // [1f, 3f];
var vector3 = new Vector3(1f, 2f, 3f).Get(2, 0, 1); // [3f, 1f, 2f];
```

GetXX and GetXXX - allows you to select 2 or 3 components of the original vector and their order by the method name, thereby forming a new two- or three-dimensional vector and returning it. It has many overloads.

```
var vector2 = new Vector3(1f, 2f, 3f).GetXZ(); // [1f, 3f];
var vector3 = new Vector3(1f, 2f, 3f).GetZXY(); // [3f, 1f, 2f];
```

InsertX - substitutes a value in the X position, thereby expanding the current vector to a four-dimensional vector and returning it as the result. Instead of X you should use X, Y, Z or W.

```
Vector3.zero.InsertX(1f); // [1, 0, 0, 0]
Vector3.zero.InsertY(1f); // [0, 1, 0, 0]
Vector3.zero.InsertZ(1f); // [0, 0, 1, 0]
Vector3.zero.InsertW(1f); // [0, 0, 0, 1]
```

MaxComponent and MinComponent - return the maximal and minimal component of the vector, respectively.

```
var max = new Vector3(3.14f, 7f, 12f).MaxComponent(); // will return 12
var min = new Vector3(3.14f, 7f, 12f).MinComponent(); // will return 3.14
```

Vector4.

With - replaces the values in the specified vector axes and returns the result as a copy. Has 3 overloads.

```
Vector4.zero.With(0, 1f);
Vector4.zero.With(0, 1f, 2, 3.14f);
Vector4.zero.With(0, 1f, 2, 3.14f, 1, 2.17f);
```

WithX, WithXX and WithXXX - substitute values in the selected vector axes and return the result as a copy. Instead of X substitute X, Y, Z or W.

```
Vector4.zero.WithX(1f);
Vector4.zero.WithXZ(0f, 2f);
Vector4.zero.WithXZW(1f, 2f, 3f);
```

Get - allows you to select 2, 3 or 4 components of a source vector and their order, thereby forming a new two-dimensional, three-dimensional or four-dimensional vector and returning it. It has 3 overloads.

```

var vector2 = new Vector4(1f, 2f, 3f, 4f).Get(0, 2); // [1f, 3f];
var vector3 = new Vector4(1f, 2f, 3f, 4f).Get(2, 0, 3); // [3f, 1f, 4f];
var vector4 = new Vector4(1f, 2f, 3f, 4f).Get(2, 0, 3, 1); // [3f, 1f, 4f, 2f];

```

GetXX, GetXXX and GetXXXXXX - allows you to select 2, 3 or 4 components of a source vector and their order by method name, thereby forming a new two- or three-dimensional vector and returning it. It has many overloads.

```

var vector2 = new Vector4(1f, 2f, 3f, 4f).GetXZ(); // [1f, 3f];
var vector3 = new Vector4(1f, 2f, 3f, 4f).GetZXW(); // [3f, 1f, 4f];
var vector4 = new Vector4(1f, 2f, 3f, 4f).GetZXWY(); // [3f, 1f, 4f, 2f];

```

MaxComponent and MinComponent return the maximum and minimum components of the vector, respectively.

```

var max = new Vector4(3.14f, 7f, 12f, -2f).MaxComponent(); // will return 12
var min = new Vector4(3.14f, 7f, 12f, -2f).MinComponent(); // will return -2

```