Extensions

Extensions are methods for capturing many types of data in Unity, which greatly enhances the day-to-day tasks of creating games.

Setup

To work with an extension, simply include the `Redcode.Extenesions` namespace in your own code.

```
using Redcode.Extensions;
```

Attention

> Many methods in this library have an overloads, which are not always used in this documentation.

- Transform
  - Position
  - Rotation
  - Scale
  - Reset states
  - Location in the hierarchy
  - Child objects
- Vectors
  - Component replacement
  - Vector extraction
  - Inserting a Component
  - Maximum and minimum component
  - Remapping
  - Absolute values
  - Vector Constraint
  - Component division
  - Checks
  - Closest point
  - Closest point on segment
- Quaternion
  - Component replacement
- GameObject
  - Interaction with components
- Component
  - Interaction with components
- CanvasScaler
  - Scale factor

- RectTransform
  - Positioning the element
- Rect
  - Changing properties
- Graphics
  - Element Color
- Colors
  - Channel replacement
- Camera
  - Background color
  - Lens shift
  - Pixel coordinates of the camera on the screen
  - Normalized camera coordinates on the screen
  - Sensor size
  - Sorting axis for transparent objects
  - Converting global coordinates to screen coordinates based on CanvasScaler
- RenderTexture
  - Texture Extraction
  - Writing to existing textures
- Scene
  - Finding Components in a Scene
  - Number of objects
- Float   Double
  - Remapping
  - Approximate comparison (only float)
- System.Object
  - Comparisons
- IComparable
  - Range check
- IEnumerable
  - Random elements
  - Excepting from collection
  - Loop through elements
  - Convert to string
- IEquatable
  - Comparisons
- IList
  - Retrieving elements
  - Removing elements

Transform

Position

Setting a new position along the desired axis: SetPositionX/Y/Z/XY/XZ/YZ or SetLocalPositionX/Y/Z/XY/XZ/YZ.

```
transform.SetPositionX(1f);
transform.SetLocalPositionXZ(1f, 2f);
```

Rotation

Setting a new rotation through Euler angles along the desired axis: SetEulerAnglesX/Y/Z/XY/XZ/YZ or SetLocalEulerAnglesX/Y/Z/XY/XZ/YZ.

```
transform.SetEulerAnglesX(45f);
transform.SetLocalEulerAnglesX(45f, 90f);
```

Scale

Setting a new scale along the desired axis: SetLocalScale or SetLocalScaleX/Y/Z/XY/XZ/YZ.

```
transform.SetLocalScale(2f); // Uniform scale.
transform.SetLocalScaleYZ(1f, 2f);
```

Reset state

Use the Reset method to reset (set position and rotation to zero and scale to one) the state of the transformation.

```
transform.Reset();
```

Location in the hierarchy.

Move object up/down in the hierarchy respectively: SetToPreviousSibling and SetToNextSibling.

```
transform.SetToPreviousSibling();
transform.SetToNextSibling();
```

Get the `Transform` component of the previous/next game object respectively: GetPreviousSiblingTransform and GetNextSiblingTransform.

```
transform.GetPreviousSiblingTransform();
transform.GetNextSiblingTransform();
```

Get all objects (their `Transform` components) located at the same level as the current one: GetAllSiblingObjects.

```
var siblings = transform.GetAllSiblingObjects();
```

Child objects

Get all child objects: GetChilds.

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

Get a random child object: GetRandomChild.

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

Add child objects: AddChilds.

```
var childs = new List<Transform>() { child1, child2 };
transform.AddChilds(children); // also supports array params.
```

Destroy all child objects: DestroyChilds.

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

Destroy all child objects by condition: DestroyChildsWhere.

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

Destroy child object by index: DestroyChild.

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

Destroy first child object: DestroyFirstChild.

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

Destroy the last child object: DestroyLastChild.

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

Vectors

Component replacement

Use the following methods to replace a component of any vector: WithX/Y/Z/XY/XZ/XW/YZ/YW/ZW/XYZ

```
// Assign a position with a zeroed value along the Y axis.
transform.position = otherTransform.position.WithY(0f);
```

Vector extraction

Use the following methods to extract a vector from any other vector: Get + any combination of X, Y, Z, and W.

```
varvector = GetSomeVector();

// Create and return a new vector from the [z, x, y] components
transform.position = vector.GetZXY();
```

Inserting a component

Use the following methods to insert a component into a vector (except Vector3Int and Vector4): InsertX/Y/Z/W.

```
varvector2 = GetSomeVector2();

// Insert value 1f in y-axis.
// From [x, y] Get [x, 1f, y].
transform.position = vector2.InsertY(1f);
```

Maximum and minimum component

Use the following methods to get the maximum and minimum components: MaxComponent and MinComponent.

```
varmax = transform.position.MaxComponent();
print($"Max Index: {max.index}\nMax Value: {max.value}");
```

Remapping

Use the Remap(min1, max1, min2, max2) method to remap the vector components to a different range. The min1 and max1 parameters specify the initial minimum and maximum values of the components in the vector. The min2 and max2 parameters indicate the final ones. The vector components will be recalculated according to how and how far these ranges differ.

```
// The Remap method will remap the vector [0.25, 0.5, 1] to [0.5, 1, 2],
var remapped = new Vector3(0.25f, 0.5f, 1f).Remap(0f, 1f, 0f, 2f);
```

Absolute values

The Abs method will return a vector with absolute components.

```
var absolutePos = transform.position.Abs();
```

Vector constraint

Use the Clamp and Clamp01 methods to limit the vector components to the desired values.

```
var clampedPos = transform.position.Clamp01();
```

Component division

The Devide method will divide the components of one vector into the corresponding components of another. Vector2 actually already supports this division with the `/` operator, but other types of vectors do not. For consistency, the Devide method has also been added to Vector2.

```
var devided = transform.position.Divide(new Vector3(0.5f, 2f, 4f));
```

Checks

Use the IsNan method to check if at least one component of a vector is NaN.

```
var isNaN = transform.position.IsNaN();
```

Use the IsUniform method to check if a vector is uniform.

```
var isUniform = transform.localScale.IsUniform();
```

Closest point

Use the GetClosestPoint method on any vector to get the point closest to it from the specified list.

```
// GetClosestPoint also supports array params.
var closest = transform.position.GetClosestPoint(pointsList);
print($"Point: {closest.point}\nIndex: {closest.index}");
```

Closest point on segment

Use the GetClosestPointOnSegment method on any vector to get the closest point that lies on the specified segment.

```
var closest = transform.position.GetClosestPointOnSegment(point1, point2);
target.position = closest.point;
```

Quaternion

Component replacement

Use the following methods to replace a quaternion component: WithX/Y/Z/XY/XZ/XW/YZ/YW/ZW/XYZ/

```
var rotation = transform.rotation.WithW(0f);
```

GameObject

Interaction with components

Use the GetOrAddComponent method to get an already existing component on the game object, or if it doesn't exist, create and get it.

```
var collider = gameObject.GetOrAddComponent<BoxCollider>();
```

Use TryGetComponentInChildren and TryGetComponentInParent to find a component on child or parent objects (including the current one).

```
if(gameObject.TryGetComponentInChildren<Renderer>(out Renderer renderer)) { ... }
```

Component

Interaction with components

Use the GetOrAddComponent method to get an already existing component on the current game object, or if it doesn't exist, create and get it. Since all scripts inherit from the `Component` class, you can also use the `this` keyword.

```
var collider = transform.GetOrAddComponent<BoxCollider>();
```

Use TryGetComponentInChildren and TryGetComponentInParent to find a component on child or parent objects (including the current one).

```
if(transform.TryGetComponentInChildren<Renderer>(out BoxCollider collider)) { ... }
```

CanvasScaler

Scale factor

The GetScaleFactor method will automatically calculate and return the scale factor for a `CanvasScaler` that is running in `CanvasScaler.ScaleMode.ScaleWithScreenSize` mode.

```
var factor = canvasScaler.GetScaleFactor();
```

RectTransform

Positioning the element

Use the SetSizeDeltaX/Y methods to set the size of an element.

```
rectTransform.SetSizeDeltaY(100f);
```

Use the SetAnchorMinX/Y and SetAnchorMaxX/Y methods to set the position of the anchors.

```
rectTransform.SetAnchorMaxX(1f);
```

Use the SetOffsetMinX/Y and SetOffsetMaxX/Y methods to set the offset from the edges. You can also use the SetLeft, SetTop, SetRight and SetBottom methods which do the same.

```
// Offset from bottom of anchors by 100 units.
rectTransform.SetBottom(100f);
```

Use the SetAnchoredPositionX/Y and SetAnchoredPosition3DX/Y/Z/XY/XZ/YZ methods to set the position relative to the anchors.

```
rectTransform.SetAnchoredPositionX(0f);
```

Use the SetPivotX/Y methods to set the pivot point of an element. Changing the anchor point will change the position of the element relative to the parent.

```
rectTransform.SetPivotX(0.5f);
```

Use the SetPivotOnlyX/Y methods to set the pivot point of an element without changing the position of the element relative to its parent.

```
rectTransform.SetPivotOnlyY(1f);
```

Rect

Changing properties

You can use the following methods to change (on a copy of) a property of a `Rect` object: WithCenter, WithPosition, WithHeight, WithWidth, WithMax, WithMin, WithSize, WithX, WithY, WithXMax, WithXMin, WithYMax, and WithYMin.

```
var rect = (transform as RectTransform).rect.WithHeight(400f);
```

Graphics

Element color

Set color to any channel of a graphical UI element: SetColorR/G/B/A/RG/RB/RA/GB/GA/BA/RGB/RGA/

```
image.SetColorA(0); // Set the alpha channel of the Image component's color to 0.
```

Colors

Channel replacement

Replacing the value of any color channel: WithR/G/B/A/RG/RB/RA/GB/GA/BA/RGB/RGA/RBA/GBA.

```
image.color = someColor.WithR(1f);
```

Camera

Background color

Change the background color using the following methods: SetBackgroundColorR/G/B/A/RG/RB/RA/GB/GA/BA/RGB/RGA/RBA/GBA.

```
Camera.main.SetBackgroundColorRB(1f, 1f);
```

Lens shift

Change the lens shift using the SetLensShiftX and SetLensShiftY methods.

```
Camera.main.SetLensShiftX(1f);
```

Camera pixel coordinates on the screen

Change any property of camera pixel coordinates on the screen using the following methods: SetPixelRectCenter, SetPixelRectPosition, SetPixelRectHeight, SetPixelRectWidth, SetPixelRectMax, SetPixelRectMin, SetPixelRectSize, SetPixelRectX, SetPixelRectY, SetPixelRectXMax, SetPixelRectYMax, SetPixelMectelXMin

```
Camera.main.SetPixelRectXMin(125f);
```

Normalized camera coordinates on the screen

Change any property of the normalized camera coordinates on the screen using the following methods: SetRectCenter, SetRectPosition, SetRectHeight, SetRectWidth, SetRectMax, SetRectMin, SetRectSize, SetRectX, SetRectY, SetRectXMax, SetRectYMax, SetRectXMin, SetRectYMin.

```
Camera.main.SetRectMax(new Vector2(1f, 1f));
```

Sensor size

Use the SetSensorSizeX and SetSensorSizeY methods to set the camera sensor size.

```
Camera.main.SetSensorSizeX(25f);
```

Sorting axis for transparent objects

The SetTransparencySortAxisX/Y/Z/XY/XZ/YZ methods will set the sorting axis for transparent objects.

```
Camera.main.SetTransparencySortAxisX(1f);
```

Convert global to screen coordinates based on CanvasScaler

Use WorldToScreenPoint to convert global coordinates to screen coordinates given the CanvasScaler component. The method will automatically take into account

```
var position = GetSomePosition();
var scaler = GetComponentInParent<CanvasScaler>();
var point = Camera.main.WorldToScreenPoint(position, scaler);
```

RenderTexture

Texture extraction

Use the ToTexture2D method to extract the texture from the RenderTexture.

```
var texture = renderTexture.ToTexture2D(TextureFormat.ARGB32);
```

Use the ToSprite method to extract the sprite from the RenderTexture.

```
image.sprite = renderTexture.ToSprite(TextureFormat.ARGB32);
```

Writing to existing textures

Use the WriteToTexture2D method to update an existing texture to match the image in the RenderTexture.

```
renderTexture.WriteToTexture2D(texture);
```

Use the WriteToSprite method to update an existing sprite to match the image in the RenderTexture.

```
renderTexture.WriteToSprite(sprite);
```

Scene

Finding components in the scene

Use FindObjectOfType and FindObjectsOfType to search for objects with the specified component in the scene.

```
var renderer = gameObject.scene.FindObjectOfType<SkinnedMeshRenderer>(includeInactive: true
```

Number of objects

The ObjectsCount method will return the total number of objects in the scene.

```
print(gameObject.scene.ObjectsCount());
```

Float and Double

Remapping

Use the Remap method to remap a value in one range to a value in another.

```
var x = 0.25f;
var remapped = x.Remap(0f, 1f, 0f, 2f);
```

Approximate comparison (floats only)

Floating point numbers are not recommended to be compared hard, instead it is better to use the Approximately comparison method, which eliminates possible unpredictable behavior.

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

System.Object

Comparisons

Use the EqualsToAll and EqualsToAny methods to determine whether all or at least one object in the specified list is equal, respectively.

```
var values = new int[] { 4, 1, 7, 2, 9 };
varx = Random.Range(0, 10);

if (x.EqualsToAny(values)) { ... }
```

IComparable

Range check

Use the IsBetween method on any IComparable object to determine if it is between two specified values or not. You can use more than just numbers.

```
var x = 0.5f;
var isBetween = x.IsBetween(0f, 1f);
```

IEnumerable

Random elements

Use GetRandomElement and GetRandomElements to get a random element or elements.

```
var element = list.GetRandomElement();
```

Use GetRandomElementWithProbability to get a random element based on their probabilities. For example, you have an array `words` containing 3 strings `["hello", "bye", "..."]` and let's say you want to get one of them with the following corresponding probabilities `[35%, 50%, 25% ]`. Then you can call the GetRandomElementWithProbability method on them like this:

```
var words = new string[] {"hello", "bye", "..."};
var random = words.GetRandomElementWithProbability(35f, 50f, 25f); // You can also pass arr

print($"Element: {random.element}\nIndex: {random.index}");
```

Use the Shuffled method to shuffle elements.

```
var list = new List<int>() { 0, 1, 2, 3 ,4 ,5};
var shuffled = list.shuffled(); // Will be shuffled.
```

Exception from IEnumerable

Use the Except method to exclude an object from the collection.

```
var listWithoutExcepted = list.Except(someObject);
```

Loop through elements

The ForEach method allows you to loop through the elements of a collection.

```
objects.ForEach(o => print(o.name));
```

Convert to string

Use AsString to convert any collection to a string in the format `"[1, 2, 3]"`.

```
var array = new int[] { 4, -7, 12, 1, 0 };
print(array.AsString()); // Prints to console -> [4, -7, 12, 1, 0]
```

IEquatable

Comparisons

Use the EqualsToAll and EqualsToAny methods to determine whether all or at least one object in the specified list is equal, respectively. Unlike similar extension methods for System.Object, these methods work with concrete types.

```
var values = new int[] { 4, 1, 7, 2, 9 };
varx = Random.Range(0, 10);

if (x.EqualsToAny(values)) { ... }
```

IList

Retrieving elements

Use the Pop method to pop an element from a list. The retrieved item is removed from the list. You can retrieve multiple items at a time.

```
var popped = list.Pop(4); // Fetched element at index 4.
```

Use the PopRandom and PopRandoms methods to populate a random element/s from a list. The retrieved element is removed from the list.

```
var popped = list.PopRandoms(2); // Extracted 2 random elements.
```

Use the PopRandomElementWithProbability method to extract a random element given probabilities. In the example below, 1 has a 50% chance, 2 has a 30% chance, and 3 has a 20% chance.

```
var list = new List<int>() { 1, 2, 3 };
var popped = list.PopRandomElementWithProbability(50f, 30f, 20f);

print($"Element: {popped.element}\nIndex: {popped.index}");
```

Removing elements

Use the RemoveRange method to remove all elements starting at the specified index.

```
list.RemoveRange(3); // Delete from index 3 to the end.
```

Good luck!