UI

Image Helper

ImageHelper:

Static Systems Package > UI > Image Helper.cs > ImageHelper.

A public static class containing a variety of functions built to make performing operations on the Image Component easier, cleaner and more efficient.

Dependencies:

UnityEngine.UI
Queuing Helper.cs
Coroutine Runner.cs

Change():

Queuing ID: **“IMAGE”**

Description:
Changes the value of Sprite within a target Image Component.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class ImageHelperTest : MonoBehaviour
{
	// Variables to pass through Change()
	Image graphic;
	Sprite newSprite;
	float delay = 0.2f;

	public void Test()
	{
		graphic = GetComponent<Image>();

		// Change the Sprite of graphic after 0.2s delay
		ImageHelper.Change(graphic, newSprite, delay);
```

```
            }
    }
```

Properties:

Type	***Property***	***Description***
Image	graphic	The target Image Component
Sprite	newSprite	The applicable Sprite
float	delay	(Optional) Delay before operation

Clear():

Queuing ID: **“IMAGE”**

Description:
Resets both the Color and Sprite of a target Image Component.
Sprite is set to a default white square and Color is set to a fully transparent white. (1f, 1f, 1f, 0f)

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class ImageHelperTest : MonoBehaviour
{
	// Variables to pass through Clear()
	Image graphic;
	float delay = 0.2f;

	public void Test()
	{
		graphic = GetComponent<Image>();

		// Reset graphic’s Sprite and Color after 0.2s delay
		ImageHelper.Clear(graphic, delay);
	}
}
```

Properties:

Type	***Property***	***Description***
Image	graphic	The target Image Component
float	delay	(Optional) Delay before operation

Recolor():

Queuing ID: **“COLOR”**

Description:
Changes the Color of a target Image Component.

Recolor() supports instant color ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth fade effect.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class ImageHelperTest : MonoBehaviour
{
      // Variables to pass through Recolor()
      Image graphic;
      Color color = new Color(1f, 0f, 0f, 1f); // Solid Red

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;


      public void Test()
      {
```

```
            graphic = GetComponent<Image>();

            // Gradually interpolate to Solid Red after 0.2s delay
            ImageHelper.Recolor(graphic, color, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Image	graphic	The target Image Component
Color	color	The applicable Color value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

Invert():

Queuing ID: **“IMAGE”**

Description:
Inverts a target Image Component.
Effects the target Image Object’s Scale, setting it to its negative/positive counterpart.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class ImageHelperTest : MonoBehaviour
{
	// Variables to pass through Invert()
	Image graphic;
	bool x = true;
	bool y = false;

	float delay = 0.2f;


	public void Test()
	{
		graphic = GetComponent<Image>();

		// Invert on the X Axis after 0.2s delay
```

```
                ImageHelper.Invert(graphic, x, y, delay);
            }
        }
```

Properties:

Type	***Property***	***Description***
Image	graphic	The target Image Component
bool	x	Invert on the X Axis
bool	y	Invert on the Y Axis
float	delay	(Optional) Delay before operation

Fade():

Queuing ID: **“COLOR”**

Description:
Gradually interpolate the alpha value of the Color of a target Component.

Similar to Recolor() in design, but more focused on modifying the alpha value in application. Takes a floating point value for the new alpha value (Automatically clamped between 0 and 1.) and uses linear interpolation to smooth the transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class ImageHelperTest : MonoBehaviour
{
	// Variables to pass through Fade()
	Image graphic;

	float targetAlpha = 0f;
	float speed = 1f;
	float delay = 0.2f;


	public void Test()
	{
		graphic = GetComponent<Image>();
```

```
                // Gradually fade the alpha of graphic after 0.2s delay
                ImageHelper.Fade(graphic, targetAlpha, speed, delay);

            }
        }
```

Properties:

Type	***Property***	***Description***
Image	graphic	The target Image Component
float	targetAlpha	The applicable floating point alpha value
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

GetSprite():

No Queuing Support.

Description:
Returns the current Sprite assigned a target Image Component.

Return Type: Sprite

```
using UnityEngine;
using UnityEngine.UI;

public class ImageHelperTest : MonoBehaviour
{
	// Variables to pass through GetSprite()
	Image graphic;


	public void Test()
	{
		graphic = GetComponent<Image>();
		Sprite sprite;

		// Get the Sprite that is assigned to graphic
		sprite = ImageHelper.GetSprite(graphic);
	}
}
```

Properties:

Type	***Property***	***Description***
Image	graphic	The target Image Component

GetColor():

No Queuing Support.

Description:
Returns the current Color applied to a target Image Component.

Return Type: Color

```
using UnityEngine;
using UnityEngine.UI;

public class ImageHelperTest : MonoBehaviour
{
	// Variables to pass through GetColor()
	Image graphic;


	public void Test()
	{
		graphic = GetComponent<Image>();
		Color color;

		// Get the Color that is assigned to graphic
		color = ImageHelper.GetColor(graphic);
	}
}
```

Properties:

Type	***Property***	***Description***
Image	graphic	The target Image Component

Slider Helper

SliderHelper:

Static Systems Package > UI > Slider Helper.cs > SliderHelper.

A public static class containing a variety of functions built to make performing operations on the Slider Component easier, cleaner and more efficient.

Dependencies:

UnityEngine.UI
Queuing Helper.cs
Coroutine Runner.cs

SetValue():

Queuing ID: **“VALUE”**

Description:
Changes the value of target Slider Component.

SetValue() supports instant value ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class SliderHelperTest : MonoBehaviour
{
	// Variables to pass through SetValue()
	Slider slider;
	float value = 1f;

	bool gradual = true;
	float speed = 1f;

	float delay = 0.2f;


	public void Test()
	{
		slider = GetComponent<Slider>();

		// Gradually change the value of the slider after 0.2s
delay
		SliderHelper.SetValue(slider, value, gradual, speed,
delay);
	}
}
```

Properties:

Type	***Property***	***Description***
Slider	slider	The target Slider Component
float	value	The applicable floating point value

bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

Toggle():

Queuing ID: **“TOGGLE”**

Description:
Changes the interactable state of a target Slider Component.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class SliderHelperTest : MonoBehaviour
{
	// Variables to pass through Toggle()
	Slider slider;
	bool toggle = false;

	float delay = 0.2f;


	public void Test()
	{
		slider = GetComponent<Slider>();

		// Changes the interactable state of slider after 0.2s
delay
		SliderHelper.Toggle(slider, toggle, delay);

	}
}
```

Properties:

Type	***Property***	***Description***
Slider	slider	The target Slider Component
bool	toggle	The applicable boolean toggle value
float	delay	(Optional) Delay before operation

SetMax():

Queuing ID: **“VALUE”**

Description:
Changes the maximum value of target Slider Component.

SetMax() supports instant value ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class SliderHelperTest : MonoBehaviour
{
	// Variables to pass through SetMax()
	Slider slider;
	float newMax = 1f;

	bool gradual = false;
	float speed = 1f;

	float delay = 0.2f;


	public void Test()
	{
		slider = GetComponent<Slider>();

		// Change the maximum value of the slider after 0.2s
delay
		SliderHelper.SetMax(slider, newMax, gradual, speed,
delay);
	}
}
```

Properties:

Type	***Property***	***Description***
Slider	slider	The target Slider Component
float	newMax	The applicable floating point maxmimum value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)

float	delay	(Optional) Delay before operation

SetMin():

Queuing ID: **“VALUE”**

Description:
Changes the minimum value of target Slider Component.

SetMin() supports instant value ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.UI;

public class SliderHelperTest : MonoBehaviour
{
	// Variables to pass through SetMin()
	Slider slider;
	float newMin = 1f;

	bool gradual = false;
	float speed = 1f;

	float delay = 0.2f;


	public void Test()
	{
		slider = GetComponent<Slider>();

		// Change the minimum value of the slider after 0.2s
delay
		SliderHelper.SetMin(slider, newMin, gradual, speed,
delay);
	}
}
```

Properties:

Type	***Property***	***Description***
Slider	slider	The target Slider Component
float	newMin	The applicable floating point minimum value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.

float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

GetValue():

No Queuing Support.

Description:
Returns the current value of a target Slider Component.

Return Type: float

```
using UnityEngine;
using UnityEngine.UI;

public class SliderManagerTest : MonoBehaviour
{
	// Variables to pass through GetValue()
	Slider slider;


	public void Test()
	{
		slider = GetComponent<Slider>();
		float value;

		// Return the current value of the slider
		value = SliderHelper.GetValue(slider);
	}
}
```

Properties:

Type	***Property***	***Description***
Slider	slider	The target Slider Component

GetMax():

No Queuing Support.

Description:
Returns the current maximum value of a target Slider Component.

Return Type: float

```
using UnityEngine;
using UnityEngine.UI;
```

```
public class SliderHelperTest : MonoBehaviour
{
	// Variables to pass through GetMax()
	Slider slider;


	public void Test()
	{
		slider = GetComponent<Slider>();
		float maxValue;

		// Return the maximum value of the slider
		maxValue = SliderHelper.GetMax(slider);
	}
}
```

Properties:

Type	***Property***	***Description***
Slider	slider	The target Slider Component

GetMin():

No Queuing Support.

Description:
Returns the current minimum value of a target Slider Component.

Return Type: float

```
using UnityEngine;
using UnityEngine.UI;
```

```
public class SliderHelperTest : MonoBehaviour
{
	// Variables to pass through GetMin()
	Slider slider;


	public void Test()
	{
		slider = GetComponent<Slider>();
		float minValue;

		// Return the minimum value of the slider
		minValue = SliderHelper.GetMin(slider);
	}
}
```

Properties:

Type	***Property***	***Description***
Slider	slider	The target Slider Component

GetToggleState():

Description:
Returns the current interactable state of a target Slider Component.

Return Type: bool

```
using UnityEngine;
using UnityEngine.UI;
```

```
public class SliderHelperTest : MonoBehaviour
{
	// Variables to pass through GetToggleState()
	Slider slider;


	public void Test()
	{
		slider = GetComponent<Slider>();
		bool toggled;

		// Get the current interactable state of the slider
		toggled = SliderHelper.GetToggleState(slider);
	}
}
```

Properties:

Type	***Property***	***Description***
Slider	slider	The target Slider Component

Text Helper

TextHelper:

Static Systems Package > UI > Text Helper.cs > TextHelper.

A public static class containing a variety of functions built to make performing operations on the TextMeshProUGUI Component easier, cleaner and more efficient.

Dependencies:

TMPro
Queuing Helper.cs
Coroutine Runner.cs

Change():

Queuing ID: **“TEXT”**

Description:
Changes the text value of a target TextMeshPro Text Component.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.TMPro;

public class TextHelperTest : MonoBehaviour
{
	// Variables to pass through Change()
	TextMeshProUGUI tmpText;
	string message = "Hello World";
	float delay = 0.2f;

	public void Test()
	{
		tmpText = GetComponent<TextMeshProUGUI>();
```

```
                // Change the Text value of tmpText after 0.2s delay
                TextHelper.Change(tmpText, message, delay);
            }
        }
```

Properties:

Type	***Property***	***Description***
TextMeshProUGUI	tmpText	The target TextMeshPro Text Component
string	message	The applicable string text value
float	delay	(Optional) Delay before operation

Clear():

Queuing ID: **“TEXT”**

Description:
Clears the text value of a target TextMeshPro Text Component.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.TMPro;

public class TextHelperTest : MonoBehaviour
{
	// Variables to pass through Clear()
	TextMeshProUGUI tmpText;
	float delay = 0.2f;

	public void Test()
	{
		tmpText = GetComponent<TextMeshProUGUI>();

		// Clear the Text value of tmpText after 0.2s delay
		TextHelper.Clear(tmpText, delay);
	}
}
```

Properties:

Type	***Property***	***Description***
TextMeshProUGUI	tmpText	The target TextMeshPro Text Component
float	delay	(Optional) Delay before operation

Recolor():

Queuing ID: **“COLOR”**

Description:
Changes the Color of a target TextMeshPro Text Component.

Recolor() supports instant color ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.TMPro;

public class TextHelperTest : MonoBehaviour
{
      // Variables to pass through Recolor()
      TextMeshProUGUI tmpText;
      Color color = new Color(1f, 0f, 0f, 1f); // Solid Red

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            tmpText = GetComponent<TextMeshProUGUI>();

            // Gradually interpolate to Solid Red after 0.2s delay
            TextHelper.Recolor(graphic, color, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
TextMeshProUGUI	tmpText	The target TextMeshPro Text Component
Color	color	The applicable Color value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

Fade():

Queuing ID: **“COLOR”**

Description:
Gradually interpolate the alpha value of the Color of a target TextMeshPro Text Component.

Fade() supports instant alpha ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Similar to Recolor() in design, but more focused on modifying the alpha value in application. Takes a floating point value for the new alpha value (Automatically clamped between 0 and 1.) and uses linear interpolation to smooth the transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.TMPro;

public class TextHelperTest : MonoBehaviour
{
      // Variables to pass through Fade()
      TextMeshProUGUI tmpText;

      float alpha = 0f;
      float speed = 1f;
      float delay = 0.2f;


      public void Test()
```

```
        {
            tmpText = GetComponent<TextMeshProUGUI>();

            // Gradually interpolate to alpha after 0.2s delay
            TextHelper.Fade(tmpText, alpha, speed, delay);

        }
    }
```

Properties:

Type	***Property***	***Description***
TextMeshProUGUI	tmpText	The target TextMeshPro Text Component
float	alpha	The applicable floating point alpha value
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

Typewrite():

Queuing ID: **“TEXT”**

Description:
Change the text value of a target TextMeshPro Text Component and display it with a typing effect.
Can take a value for charsPerSecond allowing for customizable character display rates.

Typewrite() also supports TextMeshPro Rich Text Tags.

Supports a floating point delay for timed operations.

```
using UnityEngine;
using UnityEngine.TMPro;

public class TextHelperTest : MonoBehaviour
{
	// Variables to pass through Typewrite()
	TextMeshProUGUI tmpText;
	string message = "Hello World";
	float CPS = 1f;
	float delay = 0.2f;

	public void Test()
	{
		tmpText = GetComponent<TextMeshProUGUI>();

		// Change the Text value of tmpText after 0.2s delay
		TextHelper.Typewrite(tmpText, message, CPS, delay);

	}
}
```

Properties:

Type	***Property***	***Description***
TextMeshProUGUI	tmpText	The target TextMeshPro Text Component
string	message	The applicable string text value
float	charsPerSecond	Rate of characters being written
float	delay	(Optional) Delay before operation

NumberDisplay():

Queuing ID: **“TEXT”**

Description:
Display an increasing/decreasing numerical value on a target TextMeshPro Text Component.

NumberDisplay() uses linear interpolation to smoothly blend the transition between startNum and endNum and includes a decimal clamp functionality. (0 must be used for clamping)

Supports a floating point delay for timed operations.

```
    using UnityEngine;
    using UnityEngine.TMPro;

public class TextHelperTest : MonoBehaviour
{
    // Variables to pass through NumberDisplay()
    TextMeshProUGUI tmpText;

    float startNum = 0f;
    float endNum = 100f;
    float speed = 1f;

    float delay = 0.2f;


    public void Test()
    {
```

```
        tmpText = GetComponent<TextMeshProUGUI>();

        // Display increasing/decreasing number after 0.2s delay
        TextHelper.NumberDisplay(tmpText, startNum, endNum, clamp, speed, delay);
    }
}
```

Properties:

Type	***Property***	***Description***
TextMeshProUGUI	tmpText	The target TextMeshPro Text Component
float	startNum	The applicable floating point for the starting numerical value
float	endNum	The applicable floating point for the final numerical value
float	clamp	Decimal place clamping. (E.g. 0, 0.0, 0.00)
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

GetText():

No Queuing Support.

Description:
Returns the text value of a target TextMeshPro Text Component.

Return Type: string

```
using UnityEngine;
using UnityEngine.TMPro;

public class TextHelperTest : MonoBehaviour
{
	// Variables to pass through GetText()
	TextMeshProUGUI tmpText;


	public void Test()
	{
		tmpText = GetComponent<TextMeshProUGUI>();
		string message;

		// Get the text value of tmpText
		message = TextHelper.GetText(tmpText);
	}
}
```

Properties:

Type	***Property***	***Description***
TextMeshProUGUI	tmpText	The target TextMeshPro Text Component

GetColor():

No Queuing Support.

Description:
Returns the Color value of a target TextMeshPro Text Component.

Return Type: Color

```
using UnityEngine;
using UnityEngine.TMPro;

public class TextHelperTest : MonoBehaviour
{
	// Variables to pass through GetColor()
	TextMeshProUGUI tmpText;


	public void Test()
	{
		tmpText = GetComponent<TextMeshProUGUI>();
		Color color;

		// Get the color value of tmpText
		color = TextHelper.GetColor(tmpText);
	}
}
```

Properties:

Type	***Property***	***Description***
TextMeshProUGUI	tmpText	The target TextMeshPro Text Component

🚗 Physics

Rigidbody Helper

RigidbodyHelper:

Static Systems Package > Physics > Rigidbody Helper.cs > RigidbodyHelper.

A public static class containing a variety of functions built to make performing operations on the Rigidbody Component easier, cleaner and more efficient.

Dependencies:

Queuing Helper.cs
Coroutine Runner.cs

Push():

No Queuing Support.

Description:
Apply a continuous force to a target Rigidbody Component. (Using mass)

Push() uses ForceMode.Force and should be called in FixedUpdate()

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through Push()
	Rigidbody rb;
	Vector3 direction = Vector3.forward;
	float force = 1f;


	public void Test()
	{
		rb = GetComponent<Rigidbody>();
```

```
            // Apply a continuous force of 1f to rb in the forward direction
            RigidbodyHelper.Push(rb, direction, force);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
Vector3	direction	The direction in which force is applied
float	force	The strength of the force

Launch():

No Queuing Support.

Description:
Apply an impulse force to a target Rigidbody Component. (Using mass)

Launch() uses ForceMode.Impulse

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through AddForceLaunch()
	Rigidbody rb;
	Vector3 direction = Vector3.forward;
	float force = 1f;


	public void Test()
	{
		rb = GetComponent<Rigidbody>();

		// Apply an impulse force of 1f to rb in the forward direction
		RigidbodyHelper.Launch(rb, direction, force);
```

```
            }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
Vector3	direction	The direction in which force is applied
float	force	The strength of the force

Accelerate():

No Queuing Support.

Description:
Apply a continuous acceleration force to a target Rigidbody Component. (Ignoring mass)

Accelerate() uses ForceMode.Acceleration and should be called in FixedUpdate()

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through AddForceAccelerate()
	Rigidbody rb;
	Vector3 direction = Vector3.forward;
	float force = 1f;


	public void FixedUpdate()
	{
		rb = GetComponent<Rigidbody>();

		// Apply a continuous acceleration of 1f to rb in the forward direction
		RigidbodyHelper.Accelerate(rb, direction, force);
```

```
            }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
Vector3	direction	The direction in which force is applied
float	force	The strength of the force

ForceVelocity():

No Queuing Support.

Description:
Apply an instant velocity change to a target Rigidbody Component. (Ignoring mass)

ForceVelocity() uses ForceMode.VelocityChange.

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through AddForceVelocity()
	Rigidbody rb;
	Vector3 direction = Vector3.forward;
	float force = 1f;


	public void Test()
	{
		rb = GetComponent<Rigidbody>();
```

```
            // Apply an instant velocity change of 1f to rb in the forward direction
            RigidbodyHelper.ForceVelocity(rb, direction, force);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
Vector3	direction	The direction in which force is applied
float	force	The strength of the force

SetVelocity():

Queuing ID: **"VELOCITY"**

Description:
Changes the linearVelocity of a target Rigidbody Component, ignoring mass.

SetVelocity() supports instant velocity 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition between values.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
      // Variables to pass through SetVelocity()
      Rigidbody rb;
      Vector3 newVelocity;

      bool gradual = true;

      float speed = 1f;
```

```
        float delay = 0.2f;



        public void Test()
        {
            rb = GetComponent<Rigidbody>();

            // Gradually interpolate rb.linearVelocity to equal newVelocity after 0.2s delay
            RigidbodyHelper.SetVelocity(rb, newVelocity, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
Vector3	newVelocity	The applicable Vector3 value for linearVelocity
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetMass():

***Queuing ID: "*MASS"**

Description:
Changes the Mass of a target Rigidbody Component.

SetMass() supports instant mass 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition between values.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
      // Variables to pass through SetMass()
      Rigidbody rb;
      float newMass = 10f;

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            rb = GetComponent<Rigidbody>();

            // Gradually interpolate rb.mass to equal newMass after 0.2s delay
            RigidbodyHelper.SetMass(rb, newMass, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
float	newMass	The applicable floating point mass value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetLinearDamping():

***Queuing ID: "*LINEAR_DAMPING"**

Description:
Changes the linearDamping of a target Rigidbody Component.

SetLinearDamping() supports instant damping 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition between values.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
      // Variables to pass through SetLinearDamping()
      Rigidbody rb;
      float new_ld = 10f;

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            rb = GetComponent<Rigidbody>();

            // Gradually interpolate rb.linearDamping to equal new_ld after 0.2s delay
          RigidbodyHelper.SetLinearDamping(rb, new_ld, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
float	new_ld	The applicable floating point linear damping value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetAngularDamping():

Queuing ID: **"ANGULAR_DAMPING"**

Description:
Changes the angularDamping of a target Rigidbody Component.

SetAngularDamping() supports instant damping 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
      // Variables to pass through SetAngularDamping()
      Rigidbody rb;
      float new_ad = 10f;

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;


      public void Test()
      {
        rb = GetComponent<Rigidbody>();
```

```
        // Gradually interpolate rb.angularDamping to equal new_ad after 0.2s delay
        RigidbodyHelper.SetAngularDamping(rb,new_ad,gradual,speed,delay);
    }
}
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
float	new_ad	The applicable floating point angular damping value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

ToggleGravity():

***Queuing ID: "*GRAVITY"**

Description:
Change the active state of gravity on a target Rigidbody Component.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through ToggleGravity()
	Rigidbody rb;
	bool toggle = false;
	float delay = 0.2f;


	public void Test()
	{
		rb = GetComponent<Rigidbody>();

		// Disable gravity on rb after 0.2s delay
		RigidbodyHelper.ToggleGravity(rb, toggle, delay);
```

```
            }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
bool	toggle	The applicable boolean toggle value
float	force	The strength of the force

Constrain():

No Queuing Support.

Description:
Modify the constraints on a target Rigidbody Component.

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through Constrain()
	Rigidbody rb;

	bool posX = false;
	bool posY = false;
	bool posZ = true;

	bool rotX = true;
	bool rotY = false;
	bool rotZ = true;

	bool additive = true;
```

```
        public void Test()
        {
            rb = GetComponent<Rigidbody>();

            /* Constrain Movement on the Z axis, and rotation on the X and Y
               axis, whilst keeping already existing Constraint settings */
            RigidbodyHelper.Constrain(rb, posX, posY, posZ, rotX, rotY, rotZ, additive);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
bool	posX	Constrain movement on the X Axis
bool	posY	Constrain movement on the Y Axis
bool	posZ	Constrain movement on the Z Axis
bool	rotX	Constrain rotation on the X Axis
bool	rotY	Constrain rotation on the Y Axis
bool	rotZ	Constrain rotation on the Z Axis
bool	additive	Apply new constraints alongside existing ones, instead of overriding

GetVelocity():

No Queuing Support.

Description:
Returns the linearVelocity of a target Rigidbody Component.

Return Type: Vector3

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through GetVelocity()
	Rigidbody rb;


	public void Test()
	{
		rb = GetComponent<Rigidbody>();
		Vector3 direction = new();

		// return rb.linearVelocity
		direction = RigidbodyHelper.GetVelocity(rb);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component

GetMass():

No Queuing Support.

Description:
Returns the Mass of a target Rigidbody Component.

Return Type: float

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through GetMass()
	Rigidbody rb;


	public void Test()
	{
		rb = GetComponent<Rigidbody>();
		float mass;

		// return rb.mass
		mass = RigidbodyHelper.GetMass(rb);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component

GetDamping():

No Queuing Support.

Description:
Returns the linear/angular damping of a target Rigidbody Component.

Return Type: float

```
using UnityEngine;

public class RigidbodyHelperTest : MonoBehaviour
{
	// Variables to pass through GetMass()
	Rigidbody rb;


	public void Test()
	{
		rb = GetComponent<Rigidbody>();
		float damping_l;
		float damping_a;

		// return rb.linearDamping
		damping_l = RigidbodyHelper.GetDamping(rb, true, false);
```

```
            // return rb.angularDamping
            damping_a = RigidbodyHelper.GetDamping(rb, false, true);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
bool	linear	Return linear damping
bool	angular	(Optional) Return angular damping

Rigidbody2D Helper

Rigidbody2DHelper:

Static Systems Package > Physics > Rigidbody2D Helper.cs > Rigidbody2DHelper.

A public static class containing a variety of functions built to make performing operations on the Rigidbody2D Component easier, cleaner and more efficient.

Dependencies:

Queuing Helper.cs
Coroutine Runner.cs

Push():

No Queuing Support.

Description:
Apply a continuous force to a target Rigidbody2D Component. (Using mass)

AddForcePush() uses ForceMode.Force and should be called in FixedUpdate()

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
	// Variables to pass through Push()
	Rigidbody2D rb;
	Vector2 direction = Vector2.right;
	float force = 1f;


	public void Test()
	{
		rb = GetComponent<Rigidbody2D>();
```

```
            // Apply a continuous force of 1f to rb in the forward direction
            Rigidbody2DHelper.Push(rb, direction, force);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component
Vector2	direction	The direction in which force is applied
float	force	The strength of the force

Launch():

No Queuing Support.

Description:
Apply an impulse force to a target Rigidbody2D Component. (Using mass)

AddForceLaunch() uses ForceMode.Impulse.

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
	// Variables to pass through Launch()
	Rigidbody rb;
	Vector2 direction = Vector2.right;
	float force = 1f;


	public void Test()
	{
		rb = GetComponent<Rigidbody2D>();

		// Apply an impulse force of 1f to rb in the forward direction
		Rigidbody2DHelper.Launch(rb, direction, force);
```

```
            }
        }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component
Vector2	direction	The direction in which force is applied
float	force	The strength of the force

SetVelocity():

***Queuing ID: "*VELOCITY"**

Description:
Changes the linearVelocity of a target Rigidbody2D Component, ignoring mass.

SetVelocity() supports instant velocity 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
      // Variables to pass through SetVelocity()
      Rigidbody2D rb;
      Vector2 newVelocity;

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            rb = GetComponent<Rigidbody2D>();

            // Gradually interpolate rb.linearVelocity to equal newVelocity after 0.2s delay
            Rigidbody2DHelper.SetVelocity(rb, newVelocity, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component
Vector2	newVelocity	The applicable Vector2 value for linearVelocity
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetMass():

***Queuing ID: "*MASS"**

Description:
Changes the Mass of a target Rigidbody2D Component.

SetMass() supports instant mass 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
      // Variables to pass through SetMass()
      Rigidbody2D rb;
      float newMass = 10f;

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            rb = GetComponent<Rigidbody2D>();

            // Gradually interpolate rb.mass to equal newMass after 0.2s delay
            Rigidbody2DHelper.SetMass(rb, newMass, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component
float	newMass	The applicable floating point mass value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetLinearDamping():

***Queuing ID: "*LINEAR_DAMPING"**

Description:
Changes the linearDamping of a target Rigidbody2D Component.

SetLinearDamping() supports instant damping 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
	// Variables to pass through SetLinearDamping()
	Rigidbody rb;
	float new_ld = 10f;

	bool gradual = true;

	float speed = 1f;
	float delay = 0.2f;
```

```
        public void Test()
        {
            rb = GetComponent<Rigidbody2D>();

            // Gradually interpolate rb.linearDamping to equal new_ld after 0.2s delay
            Rigidbody2DHelper.SetLinearDamping(rb, new_ld, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component
float	new_ld	The applicable floating point linear damping value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetAngularDamping():

Queuing ID: **“ANGULAR_DAMPING”**

Description:
Changes the angularDamping of a target Rigidbody2D Component.

SetAngularDamping() supports instant damping ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
      // Variables to pass through SetAngularDamping()
      Rigidbody rb;
      float new_ad = 10f;

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            rb = GetComponent<Rigidbody2D>();

            // Gradually interpolate rb.angularDamping to equal new_ad after 0.2s delay
            Rigidbody2DHelper.SetAngularDamping(rb, new_ad, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody	rb	The target Rigidbody Component
float	new_ad	The applicable floating point angular damping value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetGravity():

Queuing ID: **“GRAVITY”**

Description:
Changes the gravityScale of a target Rigidbody2D Component.

SetGravity() supports instant damping ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
      // Variables to pass through SetGravity()
      Rigidbody rb;
      float newGravity = 10f;

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            rb = GetComponent<Rigidbody2D>();

            // Gradually interpolate rb.gravityScale to equal newGravity after 0.2s delay
            Rigidbody2DHelper.SetGravity(rb, newGravity, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component
float	newGravity	The applicable floating point gravity scale value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

Constrain():

No Queuing Support.

Description:
Change the constraints on a target Rigidbody2D Component.

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
	// Variables to pass through Constrain()
	Rigidbody2D rb;

	bool posX = false;
	bool posY = false;

	bool rotZ = true;

	bool additive = true;


	public void Test()
	{
		rb = GetComponent<Rigidbody2D>();
```

```
                /* Constrain Movement on the X and Y axis, and rotation on the Z axis, whilst keeping
already existing Constraint settings */
                Rigidbody2DHelper.Constrain(rb, posX, posY, rotZ, additive);
            }
        }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component
bool	posX	Constrain movement on the X Axis
bool	posY	Constrain movement on the Y Axis
bool	rotZ	Constrain rotation on the Z Axis
bool	additive	Apply new constraints alongside existing ones, instead of overriding

GetVelocity():

No Queuing Support.

Description:
Returns the linearVelocity of a target Rigidbody2D Component.

Return Type: Vector3

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
	// Variables to pass through GetVelocity()
	Rigidbody rb;


	public void Test()
	{
		rb = GetComponent<Rigidbody2D>();
		Vector2 direction = new();

		// return rb.linearVelocity
		direction = Rigidbody2DHelper.GetVelocity(rb);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component

GetMass():

No Queuing Support.

Description:
Returns the Mass of a target Rigidbody2D Component.

Return Type: float

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
	// Variables to pass through GetMass()
	Rigidbody2D rb;


	public void Test()
	{
		rb = GetComponent<Rigidbody2D>();
		float mass;

		// return rb.mass
		mass = Rigidbody2DHelper.GetMass(rb);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component

GetDamping():

No Queuing Support.

Description:
Returns the linear/angular damping of a target Rigidbody2D Component.

Return Type: float

```
using UnityEngine;

public class Rigidbody2DHelperTest : MonoBehaviour
{
	// Variables to pass through GetMass()
	Rigidbody2D rb;


	public void Test()
	{
		rb = GetComponent<Rigidbody2D>();
		float damping_l;
		float damping_a;

		// return rb.linearDamping
		damping_l = Rigidbody2DHelper.GetDamping(rb, true, false);
```

```
            // return rb.angularDamping
            damping_a = Rigidbody2DHelper.GetDamping(rb, false, true);
        }
    }
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component
bool	linear	Return linear damping
bool	angular	Return angular damping

GetGravity():

No Queuing Support.

Description:
Returns the gravityScale of a target Rigidbody2D Component.

Return Type: float

```
using UnityEngine;

public class Rigidbody2DManagerTest : MonoBehaviour
{
	// Variables to pass through GetMass()
	Rigidbody2D rb;


	public void Test()
	{
		rb = GetComponent<Rigidbody2D>();
		float gravity;

		// return rb.mass
		gravity = Rigidbody2DHelper.GetGravity(rb);
	}
}
```

Properties:

Type	***Property***	***Description***
Rigidbody2D	rb	The target Rigidbody2D Component

Other

Animation Helper

AnimationHelper:

Static Systems Package > Other > Animation Helper.cs > AnimationHelper.

A public static class containing a variety of functions built to make performing operations on the Animator Component easier, cleaner and more efficient.

Dependencies:

No Dependencies

Trigger():

No Queuing Support.

Description:
Updates a Trigger Parameter within a target Animator Component.

```
using UnityEngine;
```

```
public class AnimationHelperTest : MonoBehaviour
{
	// Variables to pass through Trigger()
	Animator animator;
	string trigger = "Trigger_1";


	public void Test()
	{
		animator = GetComponent<Animator>();


		// Update animation trigger
		AnimationHelper.Trigger(animator, trigger);
	}
}
```

Properties:

Type	***Property***	***Description***
Animator	animator	The target Animator Component
string	trigger	The affectable Trigger Parameter

Bool():

No Queuing Support.

Description:
Updates a Boolean Parameter within a target Animator Component.

```
using UnityEngine;


public class AnimationHelperTest : MonoBehaviour
```

```
{
    // Variables to pass through Bool()
    Animator animator;
    string trigger = "Boolean_1";
    bool value = true;


    public void Test()
    {
        animator = GetComponent<Animator>();

        // Update animation boolean
        AnimationHelper.Bool(animator, trigger, value);
    }
}
```

Properties:

Type	***Property***	***Description***
Animator	animator	The target Animator Component
string	trigger	The affectable Boolean Parameter
bool	value	The applicable boolean parameter value

Integer():

No Queuing Support.

Description:
Updates an Integer Parameter within a target Animator Component.

```
using UnityEngine;

public class AnimationHelperTest : MonoBehaviour
{
    // Variables to pass through Integer()
    Animator animator;
```

```
        string trigger = "Integer_1";
        int value = 1;


        public void Test()
        {
            animator = GetComponent<Animator>();

            // Update animation integer
            AnimationHelper.Integer(animator, trigger, value);
        }
    }
```

Properties:

Type	***Property***	***Description***
Animator	animator	The target Animator Component
string	trigger	The affectable Integer Parameter
int	value	The applicable integer parameter value

Float():

No Queuing Support.

Description:
Updates a Floating-point Parameter within a target Animator Component.

```
using UnityEngine;


public class AnimationHelperTest : MonoBehaviour
{
    // Variables to pass through Float()
```

```
        Animator animator;
        string trigger = "Float_1";
        float value = 1f;


        public void Test()
        {
            animator = GetComponent<Animator>();

            // Update animation float
            AnimationHelper.Float(animator, trigger, value);
        }
    }
```

Properties:

Type	***Property***	***Description***
Animator	animator	The target Animator Component
string	trigger	The affectable Integer Parameter
int	value	The applicable integer parameter value

Audio Helper

AudioHelper:

Static Systems Package > Other > Audio Helper.cs > AudioHelper.

A public static class containing a variety of functions built to make performing operations on the Audio Component easier, cleaner and more efficient.

Dependencies:

Queuing Helper.cs
Coroutine Runner.cs

Toggle():

Queuing ID: **"TOGGLE"**

Description:
Changes the active state of a target AudioSource Component.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class AudioHelperTest : MonoBehaviour
{
	// Variables to pass through Toggle()
	AudioSource source;
	bool toggle = true;

	float delay = 0.2f;


	public void Test()
	{
		source = GetComponent<AudioSource>();

		// Enable source after 0.2s delay
		AudioHelper.Toggle(source, toggle, delay);
	}
}
```

Properties:

Type	***Property***	***Description***
AudioSource	source	The target AudioSource Component
bool	toggle	The applicable boolean toggle value
float	delay	(Optional) Delay before operation

Loop():

No Queuing Support.

Description:
Enable/disable loop on a target AudioSource Component.

```
using UnityEngine;

public class AudioHelperTest : MonoBehaviour
{
	// Variables to pass through Loop()
	AudioSource source;
	bool toggle = true;


	public void Test()
	{
		source = GetComponent<AudioSource>();

		// Enable loop on source
		AudioHelper.Toggle(source, toggle);
	}
}
```

Properties:

Type	***Property***	***Description***
AudioSource	source	The target AudioSource Component
bool	toggle	The applicable boolean toggle value

PlaySFX():

***Queuing ID:* "SOUND"**

Description:
Play a oneshot AudioClip on a target AudioSource Component.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class AudioHelperTest : MonoBehaviour
{
	// Variables to pass through PlaySFX()
	AudioSource source;
	AudioClip clip;

	float volume = 1;
	float delay = 0.2f;


	public void Test()
	{
		source = GetComponent<AudioSource>();

		// Play SFX after 0.2s delay
		AudioHelper.PlaySFX(source, clip, volume, delay);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
AudioSource	source	The target AudioSource Component
AudioClip	clip	The applicable AudioClip SFX
float	volume	(Optional) The applicable floating point volume value, clamped between 0.1 and 1
float	delay	(Optional) Delay before operation

SetAudio():

***Queuing ID:* “SOUND”**

Description:
Change the current AudioClip assigned to a target AudioSource Component

SetAudio() supports instant audio ‘snapping’ as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class AudioHelperTest : MonoBehaviour
{
      // Variables to pass through SetAudio()
      AudioSource source;
      AudioClip clip;

      bool play = true;
      bool fadeOut = true;
      bool fadeIn = true;

      float speed = 1

      float targetVolume = 1;
      float delay = 0.2f;
```

```
        public void Test()
        {
         source = GetComponent<AudioSource>();

         // Fade audio out, change audio clip and fade audio in, after 0.2s
         delay
         AudioHelper.SetAudio(source, clip, play, fadeOut, fadeIn, speed,
         targetVolume, delay);

        }
    }
```

Properties:

Type	***Property***	***Description***
AudioSource	source	The target AudioSource Component
AudioClip	clip	The applicable AudioClip SFX
bool	play	(Optional) Start playing the new audio clip after operation
bool	fadeOut	(Optional) Use linear interpolation to decrease the volume to 0 before changing the audio clip
bool	fadeIn	(Optional) Use linear interpolation to increase the volume to a target volume after changing the audio clip
float	targetVolume	(Optional) The applicable floating point volume value,

		clamped between 0.1 and 1
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetVolume():

Queuing ID: **"SOUND"**

Description:
Changes the volume of a target Audio Source Component.

SetVolume() supports instant volume 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class AudioHelperTest : MonoBehaviour
{
      // Variables to pass through SetVolume()
      AudioSource source;
      AudioClip clip;

      bool gradual = true;
      float speed = 1f;

      float newVolume = 0f;
      float delay = 0.2f;


      public void Test()
      {
```

```
            source = GetComponent<AudioSource>();

            // Gradually decrease the volume of an AudioSource to zero
            AudioHelper.SetVolume(source, newVolume, gradual, speed, delay);


        }
    }
```

Properties:

Type	***Property***	***Description***
AudioSource	source	The target AudioSource Component
float	newVolume	The applicable floating point volume value, clamped between 0.1 and 1
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

GetVolume():

No Queuing Support.

Description:
Returns the current volume of a target Audio Source Component.

Return Type: float

```
using UnityEngine;

public class AudioHelperTest : MonoBehaviour
{
	// Variables to pass through GetVolume()
	AudioSource source;

	public void Test()
	{
		source = GetComponent<AudioSource>();
		float volume;

		// Get the volume of source
		volume = AudioHelper.GetVolume(source);
	}
}
```

Properties:

Type	***Property***	***Description***

AudioSource	source	The target AudioSource Component

GetAudio():

No Queuing Support.

Description:
Returns the current AudioClip applied to a target Audio Source Component.

Return Type: AudioClip

```
using UnityEngine;

public class AudioHelperTest : MonoBehaviour
{
	// Variables to pass through GetAudio()
	AudioSource source;

	public void Test()
	{
		source = GetComponent<AudioSource>();
		AudioClip clip;

		// Get the currently applied AudioClip from source
		clip = AudioHelper.GetVolume(source);
	}
}
```

Properties:

Type	*Property*	*Description*
AudioSource	source	The target AudioSource Component

Light Helper

LightHelper:

Static Systems Package > Other > Light Helper.cs > LightHelper.

A public static class containing a variety of functions built to make performing operations on the Light Component easier, cleaner and more efficient.

Dependencies:

Queuing Helper.cs
Coroutine Runner.cs

Recolor():

***Queuing ID: "*COLOR"**

Description:
Changes the Color of a target Light Component.

Recolor() supports instant color 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class LightHelperTest : MonoBehaviour
{
      // Variables to pass through Recolor()
      Light light;
      Color color = new Color(1f, 0f, 0f, 1f); // Solid Red

      bool gradual = true;

      float speed = 1f;
```

```
        float delay = 0.2f;



        public void Test()
        {
              light = GetComponent<Light>();

              // Gradually interpolate to Solid Red after 0.2s delay
              LightHelper.Recolor(light, color, gradual, speed, delay);
        }
  }
```

Properties:

Type	***Property***	***Description***
Light	light	The target Light Component
Color	color	The applicable Color value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetIntensity():

***Queuing ID: "*INTENSITY"**

Description:
Changes the Intensity of a target Light Component.

Recolor() supports instant intensity 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class LightHelperTest : MonoBehaviour
{
      // Variables to pass through ChangeIntensity()
      Light light;
      float intensity = 10f;

      bool gradual = true;

      float speed = 1f;
      float delay = 0.2f;


      public void Test()
      {
        light = GetComponent<Light>();
```

```
        // Gradually interpolate to Intensity 10 after 0.2s delay
        LightHelper.SetIntensity(light, intensity, gradual, speed, delay);
    }
}
```

Properties:

Type	***Property***	***Description***
Light	light	The target Light Component
float	intensity	The applicable floating point intensity value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

SetRange():

***Queuing ID: "*RANGE"**

Description:
Changes the Range of a target Light Component.

ChangeRange() supports instant range 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class LightHelperTest : MonoBehaviour
{
      // Variables to pass through ChangeRange()
      Light light;
```

```
        float range = 10f;

        bool gradual = true;

        float speed = 1f;
        float delay = 0.2f;


        public void Test()
        {
              light = GetComponent<Light>();

              // Gradually interpolate to Range 10 after 0.2s delay
              LightHelper.SetRange(light, range, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Light	light	The target Light Component
float	range	The applicable floating point range value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of

Type	***Property***	***Description***
Light	light	The target Light Component
float	range	The applicable floating point range value
		Time.deltaTime)
float	delay	(Optional) Delay before operation

Toggle():

***Queuing ID: "*TOGGLE"**

Description:
Changes the active state of a target Light Component.

```
using UnityEngine;

public class LighHelperTest : MonoBehaviour
{
	// Variables to pass through Toggle()
	Light light;
	bool toggle = true;

	float delay = 0.2f;


	public void Test()
	{
		light = GetComponent<Light>();

		// Enable light after 0.2s delay
		LightHelper.Recolor(light, toggle, delay);
	}
}
```

Properties:

Type	***Property***	***Description***
Light	light	The target Light Component
bool	toggle	The applicable boolean toggle value
float	delay	(Optional) Delay before operation

GetColor():

No Queuing Support.

Description:
Returns the current Color applied to a target Light Component.

Return Type: Color

```
using UnityEngine;

public class LightHelperTest : MonoBehaviour
{
	// Variables to pass through GetColor()
	Light light;


	public void Test()
	{
		Color color;
		light = GetComponent<Light>();

		// Get the currently applied color from light
		color = LightHelper.GetColor(light);
	}
}
```

Properties:

Type	***Property***	***Description***
Light	light	The target Light Component

GetIntensity():

No Queuing Support.

Description:
Returns the current Intensity of a target Light Component.

Return Type: float

```
using UnityEngine;

public class LightHelperTest : MonoBehaviour
{
	// Variables to pass through GetIntensity()
	Light light;


	public void Test()
	{
		float intensity;
		light = GetComponent<Light>();

		// Get the current intensity from light
		intensity = LightHelper.GetIntensity(light);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
Light	light	The target Light Component

GetRange():

No Queuing Support.

Description:
Returns the current range of a Target Light Component.

Return Type: float

```
using UnityEngine;

public class LightHelperTest : MonoBehaviour
{
	// Variables to pass through GetRange()
	Light light;


	public void Test()
	{
		float range;
		light = GetComponent<Light>();
```

```
            // Get the current range from light
            range = LightHelper.GetRange(light);
        }
    }
```

Properties:

Type	***Property***	***Description***
Light	light	The target Light Component

Transform Helper

TransformHelper:

Static Systems Package > Other > Transform Helper.cs > TransformHelper.

A public static class containing a variety of functions built to make performing operations on the Transform Component easier, cleaner and more efficient.

Dependencies:

Queuing Helper.cs
Coroutine Runner.cs

Move():

***Queuing ID: "*MOVEMENT"**

Description:
Changes the position of a target Transform Component.

Move() supports instant damping 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;


public class TransformHelperTest : MonoBehaviour
{
    // Variables to pass through Move()
    public Transform transform;
    public Vector3 position = new Vector3(0, 0, 0);

    public bool gradual = true;
```

```
        public float speed = 1f;
        public float delay = 0.2f;


        public void Test()
        {
            transform = GetComponent<Transform>();

            // Gradually interpolate transform.position to equal position after 0.2s delay
            TransformHelper.Move(transform, position, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component
Vector3	position	The applicable Vector3 position value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

Rotate():

***Queuing ID: "*ROTATION"**

Description:
Changes the rotation of a target Transform Component, taking a Vector3 value for rotation and automatically converting it into a Quaternion.

Rotate() supports instant damping 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class TransformHelperTest : MonoBehaviour
{
    // Variables to pass through Rotate()
    Transform transform;
    Vector3 rotation = new Vector3(0, 0, 0);

    bool gradual = true;

    float speed = 1f;
    float delay = 0.2f;
```

```
    public void Test()
    {
        transform = GetComponent<Transform>();

        // Gradually interpolate transform.rotation to equal rotation after 0.2s delay
        TransformHelper.Rotate(transform, rotation, gradual, speed, delay);
    }
}
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component
Vector3	rotation	The applicable Vector3 rotation value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

Resize():

Queuing ID: **"SCALE"**

Description:
Changes the localScale of a target Transform Component.

Resize() supports instant damping 'snapping' as well as gradual linear interpolation using Time.deltaTime (multiplied by a customizable speed) for a smooth transition.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class TransformHelperTest : MonoBehaviour
{
	// Variables to pass through Resize()
	Transform transform;
	Vector3 scale = new Vector3(0, 0, 0);

	bool gradual = true;

	float speed = 1f;
	float delay = 0.2f;
```

```
        public void Test()
        {
            transform = GetComponent<Transform>();

            // Gradually interpolate transform.localScale to equal scale after 0.2s delay
            TransformHelper.Resize(transform, scale, gradual, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component
Vector3	scale	The applicable Vector3 scale value
bool	gradual	(Optional) use gradual interpolation for a smooth transition.
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

MoveTarget():

***Queuing ID: "*MOVEMENT"**

Description:
Changes the position of a Transform Component to match a secondary target Transform Component.

MoveTarget() uses Vector3.Distance() to measure the difference between the current and target positions alongside a floating point tolerance margin.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class TransformHelperTest : MonoBehaviour
{
      // Variables to pass through MoveTarget()
      Transform transform;
      Transform target;

      float tolerance = 0.1f;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            transform = GetComponent<Transform>();

            // Gradually interpolate transform.position to equal target. after 0.2s delay
            TransformHelper.MoveTarget(transform, target, tolerance, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component
Transform	target	The target transform to match positions with
float	tolerance	(Optional) Allowed difference between positions, recommended 0.1
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

RotateTarget():

***Queuing ID: "*ROTATION"**

Description:
Changes the rotation of a Transform Component to match a secondary target Transform Component.

RotateTarget() uses Quaternion.Angle() to measure the difference between the current and target rotations alongside a floating point tolerance margin.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class TransformHelperTest : MonoBehaviour
{
      // Variables to pass through RotateTarget()
      Transform transform;
      Transform target;

      float tolerance = 0.1f;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            transform = GetComponent<Transform>();

            // Gradually interpolate transform.rotation to equal target. after 0.2s delay
            TransformHelper.RotateTarget(transform, target, tolerance, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component
Transform	target	The target transform to match rotation with
float	tolerance	(Optional) Allowed difference between rotations, recommended 0.1
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

ResizeTarget():

Queuing ID: **“SCALE”**

Description:
Changes the localScale of a Transform Component to match a secondary target Transform Component.

ResizeTarget() uses Vector3.Distance() to measure the difference between the current and target localScales alongside a floating point tolerance margin.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class TransformHelperTest : MonoBehaviour
{
      // Variables to pass through ResizeTarget()
      Transform transform;
      Transform target;

      float tolerance = 0.1f;

      float speed = 1f;
      float delay = 0.2f;
```

```
        public void Test()
        {
            transform = GetComponent<Transform>();

            // Gradually interpolate transform.localScale to equal target. after 0.2s delay
            TransformHelper.ResizeTarget(transform, target, tolerance, speed, delay);
        }
    }
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component
Transform	target	The target transform to match positions with
float	tolerance	(Optional) Allowed difference between positions, recommended 0.1
float	speed	(Optional) Speed of interpolation. (multiplier of Time.deltaTime)
float	delay	(Optional) Delay before operation

GetPosition():

No Queuing Support.

Description:
Returns the Vector3 position of a target Transform Component.

Return Type: Vector3

```
using UnityEngine;

public class TransformHelperTest : MonoBehaviour
{
	// Variables to pass through GetPosition()
	Transform transform;


	public void Test()
	{
		transform = GetComponent<Transform>();
		Vector3 position = new();

		// Return transform.position
		position = TransformHelper.GetPosition(transform);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component

GetRotation():

No Queuing Support.

Description:
Returns the Quaternion rotation of a target Transform Component.

Return Type: Quaternion

```
using UnityEngine;

public class TransformHelperTest : MonoBehaviour
{
	// Variables to pass through GetRotation()
	Transform transform;


	public void Test()
	{
		transform = GetComponent<Transform>();
		Quaternion rotation = new();

		// Return transform.rotation
		rotation = TransformHelper.GetRotation(transform);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component

GetScale():

No Queuing Support.

Description:
Returns the Vector3 localScale of a target Transform Component.

Return Type: Vector3

```
using UnityEngine;

public class TransformHelperTest : MonoBehaviour
{
	// Variables to pass through GetScale()
	Transform transform;


	public void Test()
	{
		transform = GetComponent<Transform>();
		Vector3 scale = new();

		// Return transform.localScale
		scale = TransformHelper.GetScale(transform);
	}
```

```
    }
```

Properties:

Type	***Property***	***Description***
Transform	transform	The target Transform Component

Scene Helper

SceneHelper:

Static Systems Package > Other > Scene Helper.cs > SceneHelper.

A public static class containing a variety of functions built to make performing operations on Unity Engine Scenes easier, cleaner and more efficient.

Dependencies:

No Dependencies.

Change():

No Queuing Support.

Description:
Load a new target Unity Scene.

Supports a floating point delay for timed operations.

```
using UnityEngine;

public class SceneHelperTest : MonoBehaviour
{
	// Variables to pass through Change()
	string sceneID = "Scene_1";
	bool additive = false;

	float delay = 0.2f;

	public void Test()
	{
		// Load a new scene after an 0.2s delay
		SceneHelperz.Change(sceneID, additive, delay);
	}
}
```

Properties:

Type	***Property***	***Description***
string	sceneID	The name of the scene to load
bool	additive	Load scene as additive.
float	delay	(Optional) Delay before operation

⏰ Queuing Helper

QueuingHelper:

Static Systems Package > Technical > Queuing Helper.cs > QueuingHelper.

A public static class containing a variety of functions built to prevent coroutines from conflicting through tracking and queuing.

Dependencies:

UnityEngine.UI
Coroutine Runner.cs

CanRunCoroutine():

Description:
Checks the currently running coroutines on a specific GameObject for a coroutine with a matching Queuing ID, returns false if it finds one and sends the passed coroutine to be queued, returns true if it doesn't find any, adding the coroutine to the list of currently running coroutines. (Note that this does not run the coroutines, you will need to execute them yourself.)

Return Type: Boolean.

```
using UnityEngine;
using System.Collections;

public class QueuingHelperTest: MonoBehaviour
{
   private static void CheckQueue(IEnumerator coroutine,GameObject
key,string ID)
    {
        if(coroutine == null)
            return;

        if(QueuingHelper.CanRunCoroutine(key, coroutine, ID))
        {
            CoroutineRunner.RunCoroutine(coroutine);
        }
    }

     //. . .
}
```

Properties:

Type	***Property***	***Description***
GameObject	affectedObject	GameObject the component is assigned to.
IEnumerator	coroutine	The coroutine being checked, including all passes.
string	ID	The Queuing ID of the Coroutine. (Coroutines with matching IDs cannot run at the same time.)

FinishCoroutine():

Description:
Removes the currently running coroutine from current coroutines and runs the next coroutine of same GameObject and ID from within the queue.

Has three out variables, an IEnumerator, GameObject and String.

Must be called at the end of a coroutine.

```
using UnityEngine;
```

```
      using System.Collections;


      public class QueuingHelperTest : MonoBehaviour
      {
         //. . .

         private static IEnumerator ClearCR(TextMeshProUGUI tmpText, float
delay)
         {
            yield return new WaitForSeconds(delay);

            //Setup:
            GameObject textObject = tmpText.gameObject;

             //Apply Clear.
             tmpText.text = "";
             Debug.Log($"({textObject.name}) text value successfully
cleared!
             (Delay: {delay})");

              //Remove coroutine and run the next coroutine from the
queue.
             QueuingHelper.FinishCoroutine(textObject, "TEXT", out
IEnumerator
             coroutine, out GameObject affectedObject, out string nextID);

             CheckQueue(coroutine, affectedObject, nextID);
    }

      }
```

Properties:

Type	***Property***	***Description***
GameObject	key	The GameObject to which the affected component is assigned.

String	ID	The Queuing ID for the current coroutine
(out) IEnumerator	coroutine	The next coroutine in the queue that can be run.
(out) GameObject	affectedObject	The GameObject to which the next affected component is assigned
(out) string	nextID	The Queuing ID of the next coroutine.

CancelCoroutine():

Description:
Cancel a currently running coroutine, stopping it at its current progress and ending all tracking.

```
using UnityEngine;
using System.Collections;

public class QueuingHelperTest : MonoBehaviour
{
```

```
            public void Test(GameObject key,string ID)
            {
              // Cancel coroutine of matching GameObject and ID, running
the next
                  in the queue.
              QueuingHelper.CancelCoroutine(key, ID, true);
            }
      }
```

Properties:

Type	***Property***	***Description***
GameObject	object_key	The GameObject to which the affected component is assigned.
String	string_key	The Queuing ID for the current coroutine
bool	run_queue	Run the next coroutine in the queue?

RemoveCoroutine():

Description:
Removes a coroutine of certain GameObject, ID and Index from the queue.

```
using UnityEngine;
using System.Collections;

public class QueuingHelperTest : MonoBehaviour
{
```

```
            public void Test(GameObject key,string ID)
            {
              // Remove coroutine of matching GameObject, ID and index
from the
                  queue
              QueuingHelper.RemoveCoroutine(key, ID, 0);
            }
      }
```

Properties:

Type	***Property***	***Description***
GameObject	object_key	The GameObject to which the affected component is assigned.
String	string_key	The Queuing ID for the coroutine
int	index	The queue index.

ClearCurrentCoroutines():

Description:
Stop and clear all currently running coroutines, ending all tracking.

```
using UnityEngine;
using System.Collections;

public class QueuingHelperTest : MonoBehaviour
{
```

```
            public void Test()
            {
              // Cancel all currently running coroutines and run the next
in the
                  queue
              QueuingHelper.ClearCurrentCoroutines(true);
            }
      }
```

Properties:

Type	***Property***	***Description***
Boolean	run_queue	Run the coroutines in the queue.

ClearQueue():

Description:
Remove all coroutines from the queue.

```
using UnityEngine;
using System.Collections;

public class QueuingHelperTest : MonoBehaviour
{
```

```
            public void Test(GameObject key,string ID)
            {
              // Remove coroutine of matching GameObject, ID and index
from the
                  queue
              QueuingHelper.RemoveCoroutine(key, ID, 0);
            }
      }
```

Properties:

Type	***Property***	***Description***
None		

PrintCurrentCoroutines():

Description:
Print all the currently running coroutines to the console.

```
using UnityEngine;
using System.Collections;

public class QueuingHelperTest : MonoBehaviour
{
```

```
        public void Test()
        {
          QueuingHelper.PrintCurrentCoroutines();
        }
    }
```

Properties:

Type	*Property*	*Description*
None		

PrintQueue():

Description:
Print all the queued coroutines to the console.

```
using UnityEngine;
using System.Collections;

public class QueuingHelperTest : MonoBehaviour
{
      public void Test()
      {
        QueuingHelper.PrintQueue();
      }
}
```

Properties:

Type	***Property***	***Description***
None		